GEOPHYSICAL DATA BASE

M. R. Williamson and L. R. Kirschner

February 1975

Smithsonian Institution
Astrophysical Observatory
Cambridge, Massachusetts 02138

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont.)

ABSTRACT

This report describes a general data-management system that provides a random-access capability for large amounts of data. The system operates on a CDC 6400 computer using a combination of magnetic tape and disk storage. A Fortran subroutine package is provided to simplify the maintenance and use of the data.

GEOPHYSICAL DATA BASE

M. R. Williamson and L. R. Kirschner

1. INTRODUCTION

Our long-range objective is to design and construct a computerized geophysical data base (GDB) that will contain all available and appropriate data for the earth-dynamics areas of the Earth and Ocean Physics Applications Program. The first task was to develop a data file structure and a system for managing the data. This task is complete, and the results are described in this report. The second task, to collect and compile the data for the GDB, has been started, with a few data now available.

The GDB was designed to have maximum flexibility compatible with our immediate requirements. What has evolved is a general data-management system appropriate to a wide range of needs. The GDB provides efficient random access of many (up to disk capacity), large (up to 1.5×10^6 characters) subsets of a large (up to the number of magnetic tapes available) conglomerate of data. The system also offers file-management capabilities, including protection of the data. The GDB does not assume particular formats for data records. It could easily be a basic tool for designing a group of data sets with a more complex and specialized organization.

The GDB operates on the CDC 6400 computer. An interactive capability can be provided by the Intercom system. The GDB should be easily transferable to other modern computing systems with tape and disk facilities and a Fortran compiler. The

This work was supported in part by grant NGR 09-015-002 from the National Aeronautics and Space Administration.

GDB subroutines are written in Fortran except for some basic input/output routines that are coded in Compass. Those routines would need to be reprogramed for use on another computer.

This report serves as both a description of the GDB and a users' manual. Section 2 outlines the general capabilities and limitations of the GDB, and Sections 3 and 4 detail the data file structure and the system. Sections 5 through 7 constitute a users' manual. Structure diagrams of the GDB system organization are given in Appendix A.

2. OVERVIEW OF THE GDB

2.1 Design Considerations

A large number of data relating to the earth's lithosphere are available in computer-accessible form, and increasing amounts are expected to result from future research. Some typical current data include surface gravity, heat flow, topographic height, crustal thickness, seismic-velocity profiles, density variations, earthquake history, and plate motion. These data are mostly numerical, but future data may be of a descriptive type and may include textual information. In general, the data are complex enough to require tabular compilations.

To estimate the amount of data involved, we must determine the desired geographical resolution. The finest resolution to be cataloged must be small enough to distinguish geological detail but large enough to give a manageable number of data points. For current scientific applications, areas of 100 km × 100 km, roughly 1° × 1°, seem to satisfy these conflicting criteria best. This means approximately 4×10^4 data points. In some cases, the data will have sparse coverage over the earth's surface, while in others, the amount of data will vary significantly from point to point. Assuming an average of 80 characters per data point for each of 30 to 3000 types of data, the estimated amount of data involved is 10^8 to 10^{10} characters, a number that is consistent with Smithsonian Astrophysical Observatory's estimate of the amount of existing earth-physics data.*

Since the data will be used in many ways for several related projects, a random-access capability is needed. The critical requirement, then, is for random access of 10^8 to 10^{10} characters. This requirement dictates a dual system that uses magnetic tapes for permanent storage and disk files for temporary storage.

---

*Earth-Physics Data-Management Study, Final Report, Grant NGR 09-015-107, Smithsonian Astrophysical Observatory, Cambridge, Massachusetts, January 1971, 69 pp.

In addition, the GDB has been designed to fulfill other requirements. Owing to the large amount of data and large numbers of potential users, automatic file management is indicated – to maintain records of where and how the data are stored, to provide for protection and backup of data files, to facilitate changes or additions to the data, and to keep users informed of such changes. Because the full scope of future use cannot be predicted, it is important that there be few limitations on the structure and format of the data. The possibility of remote users suggests that the system should be made available through an interactive as well as a batch system. Finally, as is true for any system that will be in use for many years, it is desirable that the system be machine transferrable.

2.2 Components

Figure 1 is a schematic diagram of the GDB, which consists of a set of computer files and a computer system. The files are organized into a data base containing catalogs, permanent data files on disk and on magnetic tape, pool tapes, and backup tapes. The GDB system is a Fortran subroutine package. The user writes a main program, calling the GDB subroutines to manipulate the files.

Permanent GDB files are maintained as a combination of magnetic tapes and disk storage. Magnetic tapes constitute the principal means of data storage because of the large amount of data involved, but permanent direct-access data files are allowed. Two permanent disk files, the data catalog and the file catalog, contain the information necessary to access the data. The former consists of a description of each data set and includes the information used by the GDB to read the data and general information of interest to users. The file catalog comprises a description of each storage unit in the GDB, i.e., each tape or permanent disk file, and contains the necessary information for the GDB to manage data storage.

The data base looks much like an input/output device with subroutines to open and close files, read and write records, etc. Routines to create and maintain the data-base catalogs are also included. Thus, a private data base can be created by a user. Moreover, it is possible to access more than one data base from one main program.

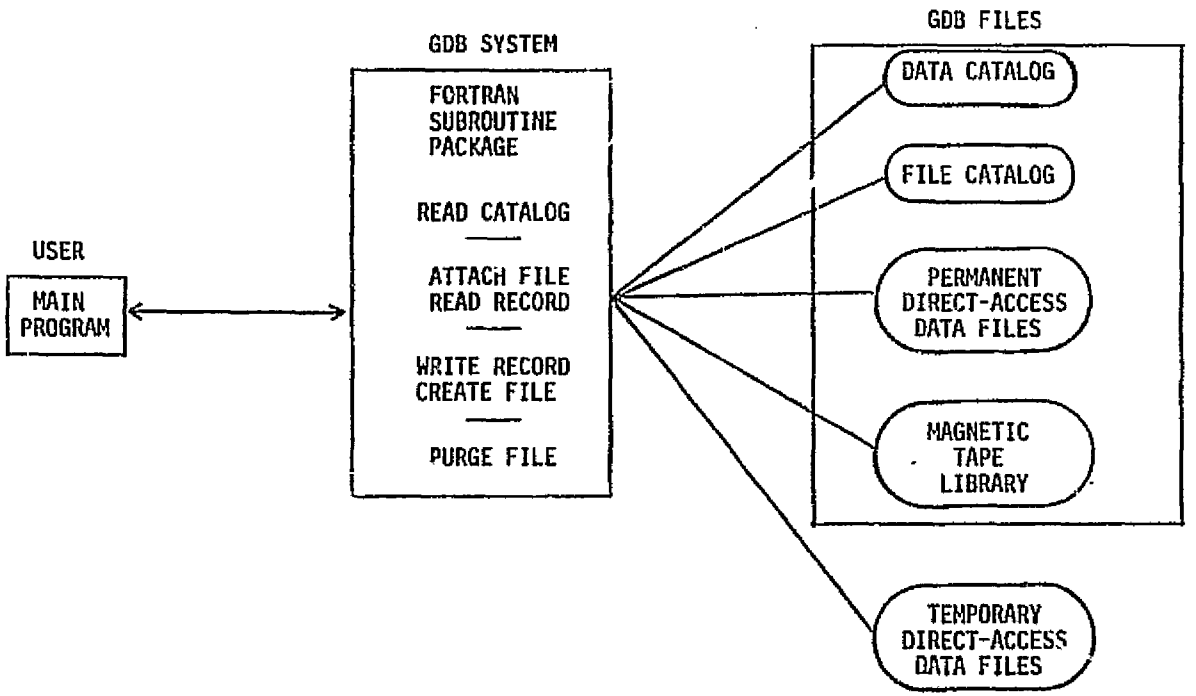


Figure 1. GDB components.

Data are not accessed directly from the magnetic tapes. Intermediate, temporary disk files are created so that the data can be accessed randomly. The GDB data files are written as one continuous string of computer words, making it possible to read or write the file with a buffer of arbitrary size. The GDB maintains an existence-bit string for each data set, which is used to locate individual records in the data file. Indirect addressing is used for data files with variable-length records. These points are discussed in detail in Section 3.1.

2.3 Capabilities and Limitations

The data in the GDB have two properties that are important to the practicality of the system. First, they are divided by type into subsets; in this report, these are called data sets. It is assumed that only a limited number of data sets will be used at the same time, the limit depending on the amount of temporary disk storage available on the computer. Second, the data sets are subdivided into ordered units of information, which we call unit records. The GDB allows random access of unit records, but there is a limit, depending on many factors, to the extent of the random access, and therefore the order of the unit records may be important.

Within that framework, the GDB allows users great flexibility in organizing and using these data sets. A data set may have fixed- or variable-length unit records; the size and number of unit records is arbitrary as long as the entire amount of data does not exceed one magnetic tape. Tapes are written without logical-record gaps, thus limiting each tape file to 1.5×10^6 characters. The number of data sets used simultaneously is arbitrary as long as the temporary disk space available is not exceeded. Since records are located by means of an existence-bit string, computer storage must allow for the bit strings when a file is being read or written. One bit for each possible record in the data set is required for every file being used. Buffers must also be supplied for all files being read or written.

The system is written so that the major elements of computer storage are provided by the user, and therefore the user retains control of the tradeoff between increased computer storage requirements and increased computer operating time. In any case, using the GDB should not significantly increase the central processor time needed to process data, but the amount of peripheral processor time can vary greatly.

The GDB provides automatic file management. The catalogs contain a complete description of the status of the data base, so users need not concern themselves with where or how the data are stored. For example, to read a data set, a user calls one subroutine to attach the catalogs, another to "attach" the data set identified by a data name and sequence number, and finally a third to return the desired unit record. Creating a data set is a similarly simple process. The GDB keeps track of all tapes automatically and provides optional backup for data sets or for the catalogs. In addition, the data catalog contains a variable-length comment section for each data set so that the data base can provide users with all the information necessary to interpret the data contained in it.

The GDB provides protection for files. All tapes in the GDB have a tape label as the first record, which must correspond to the tape description in the file catalog before it can be read or written by the GDB subroutines. In addition, there is a password system, which corresponds roughly to the permanent-file password system of the CDC 6400. The five protection passwords follow:

Read	– read any data set or catalog.
Extend	– create new data sets; add or deactivate pool tapes.
Modify	– purge any data set or catalog entry.
Control	– purge catalog.
Turnkey	– any operation.

These protection systems can be augmented by appropriate use of GDB backup procedures, which are provided for both data sets and catalogs.

The efficiency of the GDB depends largely on the user. For example, the user has control of buffer sizes; in some cases, the amount of peripheral processor time necessary to read a file will be proportional to the size of the buffer. Also, the order of the records in the file may be critical since a disk read or write is required whenever the desired record is not already in the buffer. A minimum of 7.8 min is necessary to read or write, from the disk, the largest data file allowed in the GDB (1.5×10^6 characters). On the other hand, a disk access may require 0.1 sec of peripheral processor time, and thus if a disk access were required to read or write each of 4×10^4 records, 1 hour of peripheral processor time would be needed. The tape-copy routines are as

efficient as the operating system allows. Although the peripheral processor time varies with the system environment, a full tape can be copied in as little as 5 min.

The GDB operates on the CDC 6400 computer. An interactive capability can be provided by the Intercom system. The GDB should be easily transferable to other modern computing systems with tape and disk facilities and a Fortran compiler. The GDB subroutines are written in Fortran except for some basic input/output routines that are coded in Compass. Those routines would need to be reprogramed for use on another computer.

3. FILE STRUCTURE

3.1 The Data

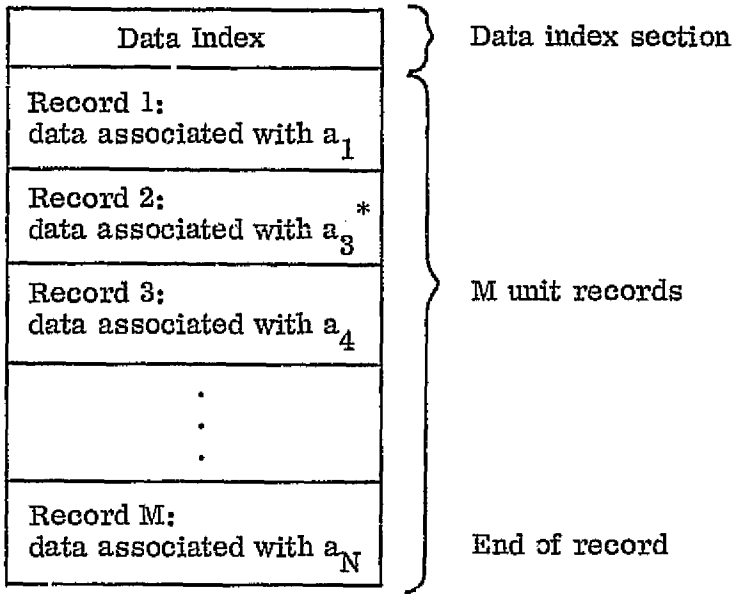
The data in the GDB are divided by type into data sets, and each set is further divided into units of information called unit records. (In this context, unit record denotes a logical division of the data. It does not mean a collection of computer words followed by an end-of-record mark.) For example, if the data set is a compilation of $1° \times 1°$ mean gravity data, each unit record might contain information such as the anomaly, the associated error, and a description of the source of the data.

A data set is identified by a seven-character data name and a sequence number, 1 to 225. Each data set is listed as one entry in the data catalog and consists of one or more tape file (or permanent disk file) copies of the data. Each copy has a unique copy number 1 to 225 and is listed as one entry in the file catalog. A tape file consists of a tape label record followed by the data file, while a permanent disk data file consists of only the data file.

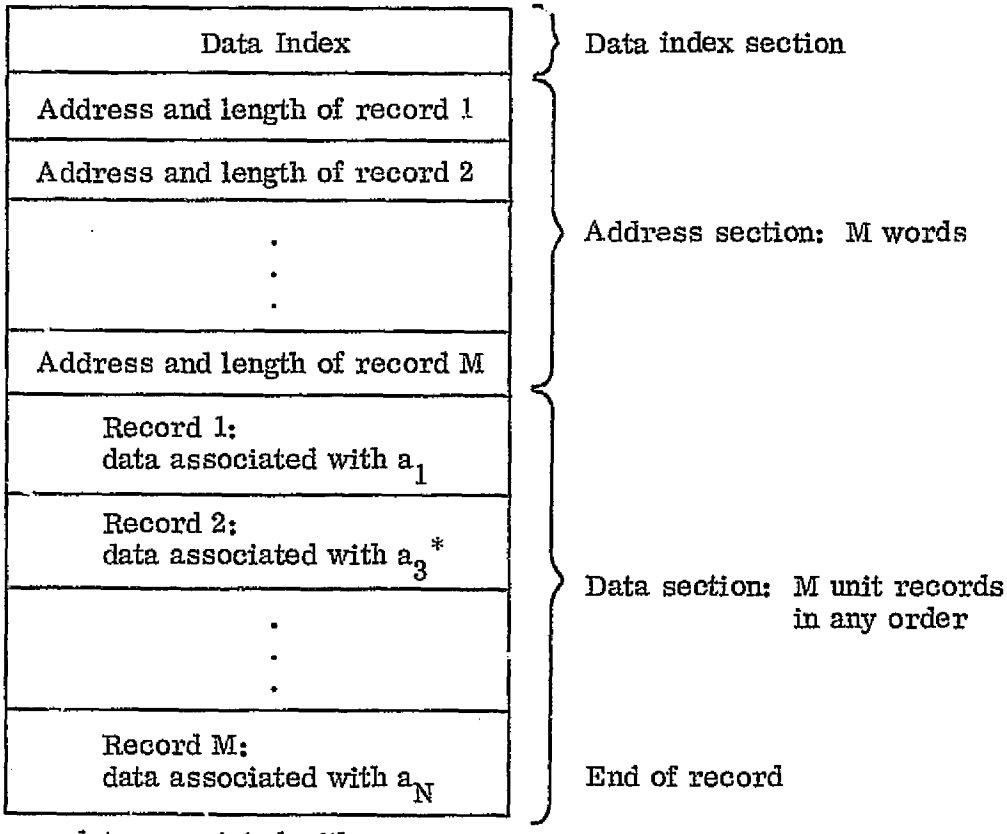
Each data file contains 1) a data index section, which is a copy of the data index record in the data catalog; 2) an address section, which is needed only if unit records have variable lengths; and 3) a data section, which contains the unit records. Figure 2 is a diagram of a data file.

Each unit record in the GDB contains the data associated with a particular area on the earth's surface, the earth being divided into unit areas ordered in some way. For data files with fixed-length unit records, the order of the records in the data section corresponds to the order of the unit areas. Missing data do not require a special code. Each data set has associated with it a string of bits, called the existence-bit string, which contains one bit for each unit area; the bit is on (or off) if there is (or is not) a unit record for that area. The GDB locates a unit record in a data file by counting on-bits in the existence-bit string. For variable-length records, an indirect addressing system uses the address section of the data file. The address

Assume that the bit string 10111100...1 (N bits; M on-bits) is associated with areas $a_1, a_2, a_3, \ldots, a_N$. Then the data file would appear as follows, with fixed-length unit records:



The data file would appear as follows, with variable-length unit records:



*Note: There are no data associated with a_2.

Figure 2. Data file.

section has one packed word for each unit record. This word contains the following pointers to the unit record in the data section:

1) The number of the physical record unit* (PRU) in the data file that contains the first word of the unit record.

2) The position in that PRU of the first word of the unit record.

3) The length of the unit record in words.

The address of the variable-length record is located by counting the existence bits. The variable-length records in the data section are in the order in which they were written by the program that created the file.

The GDB uses pool tapes, data tapes, and catalog backup tapes. The first record of all tapes is the tape label record, which is a copy of the file index record in the file catalog: For pool tapes, the tape label record is followed by an end-of-file; for data tapes, it is followed by the data file and an end-of-file; and for tapes that contain a backup copy of the data or file catalog, it is followed by a copy of the appropriate direct-access file.

3.2 The Catalogs

The data and file catalogs utilized by the GDB to manage the data provide a complete description of the status of the GDB. Each data set is described by one entry in the data catalog, and each tape and permanent disk file is described by one entry in the file catalog.

The data catalog is a permanent direct-access file consisting of variable-length records, each one of which is called a data index record. One data index record exists for each data set (data name, sequence number) in the GDB, with the following format:

1) Length of index record in words. Integer.
2) Data name. Seven characters.
3) Sequence number. Integer 1–255.
4) Creation date. Hollerith, as returned by the function ITIMER(1).
5) Maximum unit record length in words. Integer.
6) Fixed-length (= 0) or variable-length (= 1) unit records.

*The physical record unit of a device is the basic information unit for reading or writing. For the disk, it is 64 words; for magnetic tapes in binary mode, the PRU size is 512 words.

7) Number of unit records. Integer.
8) Number of disk PRUs in the index section. Integer.
9) Number of PRUs in the address section. Integer.
10) Number of PRUs in the data section. Integer.
11) Number of existence bits. Integer.
12) Length of comments in words. Integer.
13) ... (n) existence-bit array.
n+1) ... (m) comments including a description of the data, format, and sources.

The data index record is also described in Section 6.1. The first record in the data catalog contains the following:

1) Length = 12.
2) Data name = 7RDATACAT.
3) Sequence number of the data catalog.
4) Creation date of the data catalog.
5) Maximum length of the data index records.
6) 1.
7) Number of data index records (including the special data-catalog index record) that are in the data catalog.
8) 0.
9) 0.
10) 0.
11) 0.
12) 0.

The data catalog is sorted by data name and sequence number.

The file catalog is a permanent direct-access file consisting of fixed-length records, each one of which is called a file index record. There is one file index record for each tape and permanent direct-access file associated with the GDB. Each file index record is eight words long and has the following format:

1) File device, tape (=0) or permanent direct access (=1).
2) File identification (reel number or permanent file name). 10 characters.
3) Data name. Seven characters.
4) Sequence number. Integer 1–255.
5) Copy number. Integer 1–255.
6) Creation date. Hollerith, as returned by the function ITIMER(1).

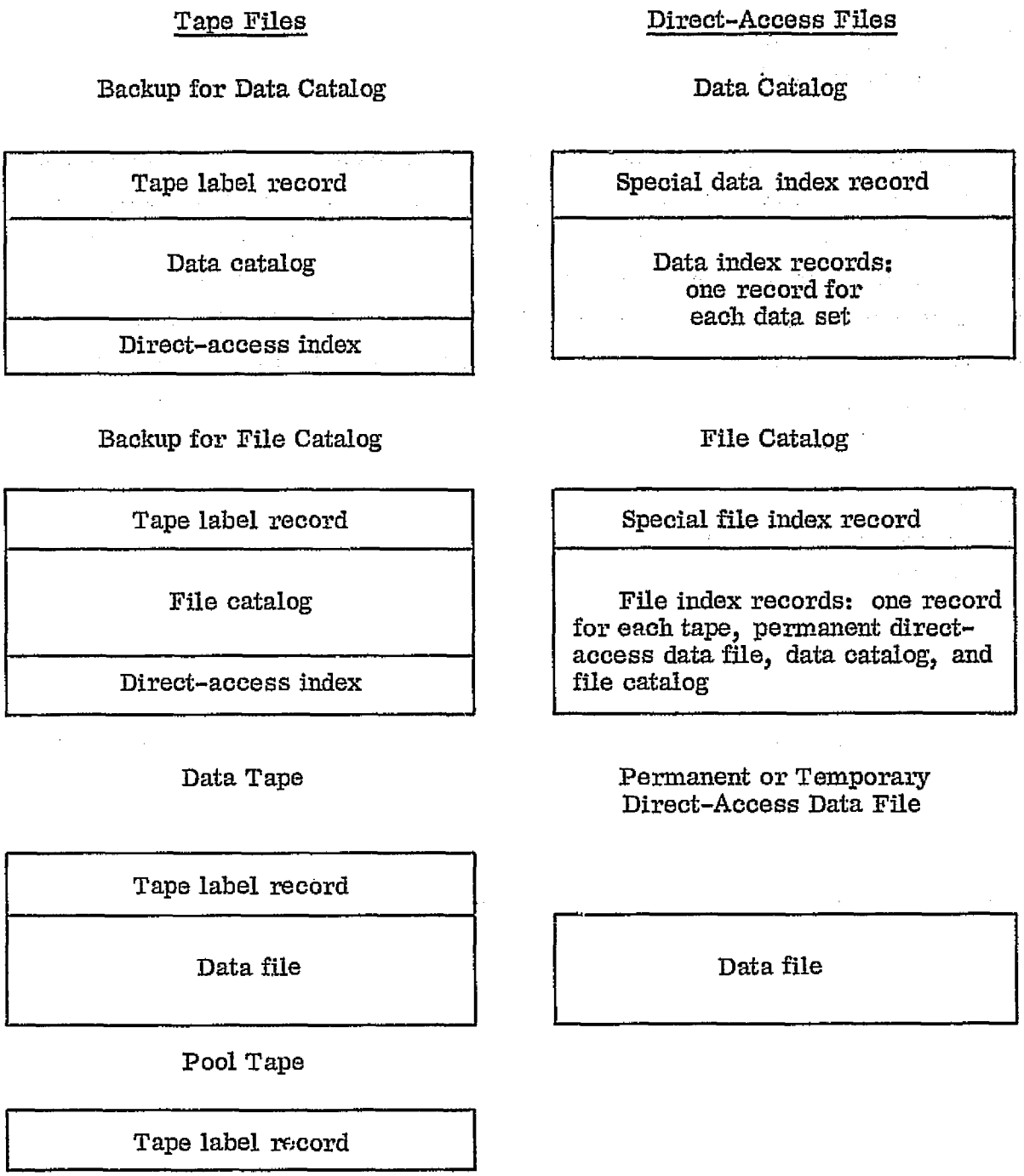
7) Tape density. Integer.

8) Length of tape in feet. Integer.

The data name and sequence number of a data set are as given in the data catalog. For backup copies of either the data or the file catalog, the data name is 7RDATACAT or 7RFILECAT, while for pool tapes, the name is specified by the owner of the tape. For tapes, the file identification is the tape number (the visual reel number), i.e., the number printed on the tape. For permanent direct-access files, the file identification is the permanent-file name ($\leq$10 characters). The file index record is also described in Section 6.2. The first record in the file catalog is the special file catalog index record, with the following format:

1) 1.
2) Permanent-file name of file catalog.
3) 7RFILECAT.
4) Sequence number of the file catalog.
5) Copy number of the file catalog.
6) Creation date of the file catalog.
7) Number of file index records in the file catalog, including the special file-catalog index record.
8) 0.

Figure 3 is a diagram of the file organization.



File index record = Tape label record
Data index record= Data label section of data file

Figure 3. Diagram of file organization.

4. THE GDB SYSTEM

4.1 User Functions

The GDB system is a collection of Fortran subroutines called by the main program, which is written by the user. The subroutines are divided into groups corresponding to the protection passwords. Those subroutines requiring read permission only are the following:

GDOPEN – GDB Open Catalogs
GDRCAT – GDB Release Catalogs
GDADF – GDB Attach Data File
GDRDISK – GDB Return Disk File
GDREAD – GDB Read Data Record
GDRDIR – GDB Read Data Index Record
GDRFIR – GDB Read File Index Record
GDPCAT – GDB Print Catalog

To read the data in the GDB, the user calls GDOPEN, GDADF, and GDREAD. GDRDIR, GDRFIR, and GDPCAT are used to obtain information about the contents of the GDB. GDRDISK and GDRCAT release disk files from the user's program.

A data file is written as a temporary disk file without any passwords. It does not become a permanent part of the GDB until it is "created." The write subroutines follow:

GDWRIT – GDB Write Sequential Data Record
GDREWR – GDB Rewrite or Random Write

Four subroutines that are sometimes useful in conjunction with the read/write subroutines, and for which no passwords are needed, follow:

IBIT – Set Indicated Bit
NETBIT – Retrieve Bit Value
EXBIT – Set Existence Bit
LATLON – Get Latitude and Longitude

The following subroutines require both read and extend permission:

GDCDF – GDB Create Data File
GDEUP – GDB End Update
GDDEACT – GDB Deactivate Pool Tape
GDAPT – GDB Add Pool Tape

To add a data set to the GDB, the user first writes the file using GDWRIT or GDREWR and then adds it to the system by calling GDCDF and GDEUP. To alter a GDB data set, the user creates a new version and purges the old one. Creating a data set requires a GDB pool tape. GDAPT is used to add pool tapes to the GDB, and GDDEACT can be utilized to deactivate bad pool tapes.

The subroutines requiring read, extend, and modify permission as follows:

GDPURDS – GDB Purge Data Set
GDRPF – GDB Release Permanent File
GDPFIR – GDB Purge File Index Record
GDPDIR – GDB Purge Data Index Record
GDRDT – GDB Release Data Tape
GDRWDI – GDB Rewrite Data Index Record
GDRWFI – GDB Rewrite File Index Record
GDAFIR – GDB Add File Index Record
GDWDIR – GDB Write Data Index Record
GDSAVE – GDB Save a Data Tape

These are used to purge old files, maintain the GDB, and recover from system failures. Normally, GDPURDS would be called to remove a data set completely from the data base. However, abnormal situations occasionally require the use of one or more of the other subroutines.

Finally, the following subroutines require read, extend, modify, and control permission:

GDRELOD – GDB Reload Catalog
GDCREAT – GDB Create Catalog

4.2 Data Retrieval

The first step in retrieving data from the GDB is to attach the data and file catalogs. This causes the GDB to set up an array in core for each catalog. The array contains one packed word for each entry (index record) in the catalog. A maximum of 100 entries for each catalog is currently allowed. When a data set is "attached," the arrays are searched and the index records for the data set are read from the catalogs. If the copy of the data set requested is a magnetic-tape file, the tape is requested, the tape label is checked with the file index record, and the tape is copied to a temporary disk file. If the data file is a permanent disk file, the file is attached to the user's program.

The data index record is returned to the user from the data catalog when the data set is attached. The user then passes the index record to the read subroutine, thus giving the read subroutine the information necessary to retrieve unit records from the data file. The existence-bit string, included in the index record, contains one bit for each unit area and the bit is on (or off) if there is (or is not) a unit record for that unit area. The GDB locates a unit record in a data file by counting on-bits in the existence-bit string. For data files with fixed-length unit records, the record can be located directly by multiplying the length of the record by the number of preceding records (on-bits). For data files with variable-length records, the address and length of the unit record are located in the same way since the address section of the data file is like a one-word fixed-length data file. Thus, accessing a data file with variable-length records is a two-step process and may require two disk reads.

When the data are read from the disk, a full buffer of information is read. The buffer for each file is provided by the user, and the length of the buffer is arbitrary. For data files with variable-length unit records, the buffer is partitioned into one section to read the address section of the data file and a second to read the data section. The read subroutine keeps track of which information is in the buffer and accesses the data file on the disk only if the desired record is not in the buffer. If a disk read is required, the unit record can be positioned at the beginning, middle, or end of the buffer. The user has control of this positioning.

The user has two options for specifying which records are to be retrieved: The record can be identified by the number of its corresponding existence bit or by a bit string with the bit corresponding to its existence bit turned on.

4.3 Data File Creation

To write a new GDB data file, the user must either write the file sequentially in the same order as the existence-bit string or set up the existence-bit string before writing the file randomly. The user program calls the write subroutine once for each record to be written and once to terminate the write.

The sequential-write subroutine fills a buffer with the unit records and writes the buffer onto the disk whenever it is full. The subroutine leaves space at the beginning of the data file for the data index section. The user passes the data index record to the subroutine on the last call, and the data index record is written in the space reserved for it. If the data file has variable-length records, space is also reserved for the address section of the data file. The buffer is partitioned into two sections, one to write the address section of the data file and the other to write the data section.

The random-write subroutine can write a new data file or alter an existing one. If a new file is being written, blank space is reserved on the disk and the random write then changes the blank file. The subroutine always reads and writes a full buffer of information, and if the information that is to be changed is in the buffer, no disk access is necessary. For data files with variable-length records, the buffer has one section to read or write the address section and another one to read or write the data section of the data file.

The user must set the existence-bit string before calling the random-write subroutine for the first time. For data files with fixed-length records, the subroutine locates the unit record in the data file by counting on-bits in the existence-bit string. The subroutine changes the record by writing the new record over the old one. For data files with variable-length records, the address and length of the unit record are located in the address section. Then the address is changed to that of the end of the data file, and the new version of the record is written there. The old record remains in the file but is never used again. If a disk read/write is required, the record to be changed can be positioned at the beginning, middle, or end of the buffer as designated by the user.

The user makes a new data set a permanent part of the data base by calling a "create" subroutine, which catalogs the file as a permanent disk file or copies the file to magnetic tape and makes the appropriate changes in the catalogs.

4.4 Protection Mechanisms

The GDB files are protected by a tape label system, backup procedures, and a password system roughly corresponding to the CDC 6400 permanent-file password system.

The first record on all tapes in the GDB is a tape label, which is the same as the file index record in the file catalog. Whenever a tape is requested by the GDB system, its label record is checked. If the record does not correspond to the file index record, a message is written on the computer console instructing the operator to check the visual reel number of the tape. If the tape label is still incorrect after this check, an error indicator is set, a diagnostic message is printed, and control is returned to the user's program.

It is advisable to back up the catalogs within the GDB system periodically. In addition, it is wise to have more than one tape copy of any data set that is difficult to recreate. If the GDB catalogs must be recreated from backup, it is possible, for example, that a tape containing a data file will be listed in the backup file catalog as a pool tape. This tape cannot be used by the GDB system until its file catalog entry has been updated, however, because the tape label will differ from the file index record.

The GDB passwords follow:

Read — read any data set or catalog.
Extend — create new data sets; add or deactivate pool tapes.
Modify — purge any data set or catalog entry.
Control — purge catalog.
Turnkey — any operation.

The passwords are specified when the user attaches the catalogs. If the user calls a subroutine without having specified the necessary passwords for it, an error flag

is set, an error message is printed, and control is returned to the user's program without the subroutine being executed.

This protection system applies to the data base as a whole; files are not protected on an individual basis. Therefore, it is suggested that all operations requiring modify or control passwords be carried out by one person. If this is not convenient, the data base can be divided into several data bases, each managed by a single person. More than one data base can be used simultaneously. Also, it is a simple process to transfer files from one data base to another.

This system of protection has been implemented through the permanent-file password system. Any operation performed on GDB data files is preceded by one on a GDB catalog. Since the data and file catalogs are permanent files, using the permanent-file password system to protect the catalogs will, in fact, protect the data files themselves. However, to the CDC 6400 operating system, the operations necessary to create a data file appear the same as those necessary to purge one. Thus, operations requiring both extend and modify are protected by the operating system through the extend password only, and the GDB system, not the operating system, checks the modify password whenever a purge operation is done.

5. USERS' GUIDE TO THE GDB

5.1 Introduction

The GDB system is a Fortran subroutine package. The user writes a main program calling the subroutines described in Section 6. Sections 5.2 to 5.5 detail the common user operations, and Section 5.6 gives some practical information concerning data-base management.

The GDB files are protected by a password system that utilizes the following words: read, extend, modify, control, and turnkey. The GDB subroutines are listed with the passwords required. Users with only read permission cannot alter the GDB files. Read and extend permission are required to create new files. Users should inform the GDB manager whenever a "create" program has terminated abnormally. Although it is impossible for the GDB files to be destroyed, it is possible that some pool tapes will have to be relabeled by the manager. The manager can also help a user recover a new file that was lost owing to the abnormal termination of the program that created it. The data-base manager has modify, control, and turnkey permission. A user who wishes to create a private data base should read Sections 3 and 4, which are also helpful for dealing with sophisticated applications.

5.2 Read

To read data in the data base, the user writes a main program calling the GDB subroutines GDOPEN, GDADF, and GDREAD. The GDOPEN attaches the GDB catalogs to the user's program; GDADF attaches data files to the program; and GDREAD retrieves unit records from these data files. These subroutines are described in Sections 6.3, 6.4, and 6.5. Section 5.4 gives further information concerning the efficient use of GDREAD.

The first step in any run of the GDB is to call GDOPEN, thus attaching the file catalog and the data catalog, which are permanent files residing on the disk. The

user must know both the permanent-file name of the GDB file catalog and the read password.

The variable ICOND is an error flag that appears in the calling sequence of most GDB subroutines. If an error is encountered, the subroutine writes an error comment on a file specified by the user and returns control to the calling program with ICOND $\neq$ 0. An exception is the condition ICOND = 22 in subroutine GDREAD, for which no error comment is printed. The user should test ICOND after each call to any subroutine. Section 6.32 contains a table of ICOND values.

For a data file to be read, it must be "attached" by calling GDADF. The user must know the data name of the data file. By means of the subroutine GDPCAT, information can be obtained about the contents of the data files. Every file in the GDB is uniquely identified by a data name, sequence number, and copy number, the last two being integers from 1 to 255. The default condition in GDADF is to attach the file with the highest sequence number and lowest copy number. However, the user may specify particular sequence and copy numbers. GDADF returns to the user the data index record (see Section 6.1) from the data catalog. This record is input to GDREAD, but since GDREAD does not use the comment section, it is not necessary to retrieve that part of the index record in the call to GDADF.

The existence-bit string associated with the file is part of the index record. The bit string has one bit for each unit area (or key) associated with the file. The bit is on (or off) if there is (or is not) a unit record in the data file for that unit area. The GDB locates unit records by counting on-bits in the existence bit string. The user can determine whether or not a record exists by checking its corresponding existence bit, using subroutine NETBIT. If a user is not interested in reading the file, the index record with the existence-bit string should be obtained from subroutine GDRDIR, rather than GDADF, since GDRDIR does not attach the data file.

If the file to be attached is on magnetic tape, the tape is copied to a temporary disk file, and the user must provide a buffer to be used by GDADF for the copy operation. The buffer should be as large as possible and can be reused.

If many data files are to be processed sequentially, it is not necessary, and probably not advisable, to attach them all at the same time. GDRDISK will detach the data file and allow the same file name and temporary disk storage to be used for another data file.

Once the data files have been attached by calling GDADF, the catalogs are no longer needed for reading the files and can be released by calling GDRCAT. Only one set of catalogs can be attached to a program at any one time. If files from different data bases are required, the catalogs from one data base must be released by GDRCAT before those from another can be attached by GDOPEN.

Once the data files have been attached, GDREAD will return individual unit records from the files to the calling program. The user has two options for specifying which data record is to be returned: If the variable MODE = 0, then array IBIT, corresponding to the existence-bit string, is used. On each call to GDREAD, NBIT is incremented to the next on-bit in IBIT and the corresponding record is returned. If MODE = 1, on each call to GDREAD, NBIT is incremented by 1 and the record corresponding to that bit in the existence-bit string is returned. In either case, if the designated record does not exist, GDREAD returns with ICOND = 22.

This dual-read system suggests two obvious modes of use. For the "automatic" mode, the user sets up an IBIT array, sets MODE = 0, initializes NBIT = 0, and calls GDREAD once for each record specified by IBIT. In the "manual" mode, the user sets MODE = 1 and then uses NBIT to specify which record is desired. The automatic mode results in a sequential read of the data file, while the manual mode may or may not be sequential.

The user provides GDREAD with a buffer, and a full buffer of information is always read from the disk. Thus, GDREAD does not access the disk every time it is called. In the manual mode, a large number of disk reads may be necessary. Each disk read may take 0.1 sec of peripheral processor time. It is the responsibility of the user to estimate the peripheral processor time connected with the job to ensure that it is not too large. Correct use of the variable ITYPE can reduce the peripheral processor time significantly (see Section 5.4).

5.3 Write

To write a data file, the user first writes a main program calling either GDWRIT or GDREWR once for each record to be written. The catalogs do not have to be attached in order to call GDWRIT or GDREWR.

GDWRIT writes data files sequentially. It must receive records in the order required for the output file. However, it is not necessary to specify the existence-bit array until after the file has been written. Initially, the GDWRIT subroutine uses only the length of the data index record, the data type indicator, and, if the data file has variable-length records, a maximum for the number of records to be written. After the last record has been written, the user makes a final call to GDWRIT, providing the data index record. Each record written in the data file must have the corresponding existence bit turned on (=1). The subroutine IBIT can be used to turn on the bits in the existence-bit string.

GDREWR allows users to do random-access writing of data files. However, the existence-bit array must be set up before the first call to GDREWR. The user provides GDREWR with a buffer so that GDREWR does not access the disk every time it is called. However, if the records are not at least partially ordered, GDREWR could take up to 0.2 sec of peripheral processor time for each record written. The user must estimate the peripheral processor time connected with the job to ensure that it is not too large. Correct use of the variable ITYPE can reduce that amount of time significantly (see Section 5.4).

If GDREWR is employed to write a data file with variable-length records, the resulting file will have an address section arranged in the same order as the existence-bit string and a data section arranged in the order in which it was written, a system that may or may not be satisfactory for future use. To sort such a file written by GDREWR, the file could be read with GDREAD and then rewritten sequentially by GDWRIT (or GDREWR).

With GDREWR, changes can also be made to an existing GDB file. The user must call GDOPEN to attach the catalogs, call GDADF to attach the file as a temporary disk file, and then call GDREWR for each record to be changed. For fixed-length records, the

new version of the record must be the same length as the old one. For variable-length records, the new version can be any length since it will be written at the end of the file. Note that the old version of the record remains in the file and the file will be longer than necessary. Also, the records will not physically be in order on the tape. If necessary, this situation can be corrected by rewriting the file sequentially with GDWRIT or GDREWR

To add to an existing GDB file, the file must be rewritten. The user calls GDOPEN and GDADF and then writes a new file by copying the old records and including the new records, using either GDWRIT or GDREWR.

In any case, the newly written file does not automatically become part of the data base; to become permanent, it must be entered in the catalogs, as described in Section 5.5.

5.4 Efficient Use of Read/Write

If a user wishes to read or write many large files in the GDB, efficient use of those files becomes important, with the choice of buffer sizes and of the algorithm that positions data in the buffers (ITYPE) being most important. Also, alterations in the data files or in the procedures for processing the files may be necessary.

In order to attach a data file residing on tape, the GDB copies the tape to a direct-access disk file. Likewise, to create a new data file, a disk file is copied to tape. In these cases, the I/O buffers, supplied by the user, should be as large as possible, especially if the files are large or if many files are being processed. While the exact time to copy the file varies with the system environment, the buffer size greatly affects the efficiency. Since tape is the slower device, the minimum buffer size should be two or three tape PRUs (512 words each). For optimum efficiency, the buffer size might be as high as 20K. This buffer can be reused after the data file is attached or created.

To read and write data files, the user must supply an I/O buffer that is passed to the read/write routines, the optimum length of the buffer depending on the specific application. If it is too short, I/O will be inefficient; if it is too long, buffer space may be wasted. Thus, in order to choose an appropriate buffer length, both I/O efficiency and the amount of central memory available for buffers must be considered.

The goal in choosing a buffer size for the read/write routines is to minimize the actual number of disk accesses. The GDB accesses the disk only when the desired unit record is not already in the buffer. Thus, for files being read or written sequentially, the buffer should be as large as possible. On the other hand, if the file is to be read or written randomly (i.e., the next record is so far from the last one that it cannot possibly be in the buffer), then making the buffer larger than the minimum would be a waste of space. For example,

Unit record size = 3 words
Number of unit records = 60,000

Call to GDREAD	Unit record number
1	1
2	60000
3	2
4	59999
5	3

In this case, no matter how big the buffer is, a disk access will be necessary on each call to GDREAD, so the user might as well set the buffer size to the disk PRU size of 64 words (or the next multiple of 64 words greater than the unit record size).

It is important to note that a disk access can take 0.1 sec of I/O time. Since a tape without record marks holds about 1,500,000 words of data, the effect of buffer lengths is as follows:

Buffer length (words)	Peripheral processor time to read or write (from the disk) a full tape's worth of data (min)
3200	7.8
1600	15.6
512	46.8

The peripheral processor time cannot be reduced further by using a buffer greater than a half-track (3200 words) because the disk head must be repositioned if more than half a track of data is accessed.

The user can control which information is read or written into the buffer by the variable ITYPE. The desired record can be positioned in the beginning (ITYPE = 1), end (ITYPE = 2), or middle (ITYPE = 3) of the buffer. ITYPE should reflect the order of the accessed data records in the data file:

ITYPE = 1 , forward read/write
ITYPE = 2 , reverse read/write
ITYPE = 3 , random read/write .

For example,

Subroutine call	1	2	3	4	5	6	7	8	9	10
NBIT	1	2	3	10	9	8	7	4	5	6
ITYPE	1	1	1	2	2	2	2	1	1	1

If the data records are not to be read or written in any particular order, set ITYPE = 3.

5.5 Creation of a Data Set

To create a data set, a user writes a main program calling first GDWRIT or GDREWR and then GDOPEN, GDCDF, and GDEUP. Either of the first two subroutines writes a temporary disk file, as described in Section 5.3, which must be cataloged as a permanent disk file or copied to tape in order to make it a permanent part of the data base. The catalogs must be attached by calling GDOPEN with both read and extend passwords. Then GDCDF is called to make changes to the catalogs and make the new data file permanent.

At the end of a job that creates new data sets, the user must call GDEUP to make the changes permanent by extending the catalogs, to sort and print the new catalogs, and to create backup for the catalogs. When writing or cataloging a data set, users should always check ICOND, and if ICOND $\neq$ 0, GDEUP should not be called. As long as GDEUP has not been called, no permanent changes to the GDB catalogs are made. For example, the file catalog still lists tapes written by GDCDF as pool tapes, and those tapes cannot be used until the entry in the file catalog agrees with the tape label. Such inconsistencies must be fixed by the data-base manager. However, the pool tape can be removed from active status with the subroutine GDDEACT, and new pool tapes can be added with GDAPT.

5.6 Maintenance of a Data Base

The maintenance of a data base should be the responsibility of only one person, who will perform all the operations that require modify or control passwords, including creating the data base, normal purging of data sets, and recovery from user errors or operating-system failures. Anyone who will be maintaining a data base should read Sections 3 and 4.

To set up a private data base, it is first necessary to create both an initial data catalog and an initial file catalog, accomplished by calling subroutine GDCREAT from a user-supplied main program. Once the catalogs exist as permanent files on the disk, the user can add pool tapes to the file catalog by calling GDAPT and then proceed to create data files.

It is advisable to dump the catalogs to tape periodically in case the permanent-file versions are accidentally destroyed. This is done by calling GDEUP at the end of any GDB run with the backup-copy flags turned on. If many changes are made to the catalogs during the day, it is wise to back up the catalogs with GDEUP. Reloading from data-base backup tapes involves calling subroutine GDRELOD with the visual reel numbers of the backup tapes. In reloading, note that the data sequence numbers of the data and file catalogs must always match.

The catalogs should be dumped to tape periodically by GDEUP and reloaded by GDRELOD. This procedure deletes inactive entries from the permanent disk files and therefore reduces the amount of disk space required.

Although a data set is normally removed from the data base by calling the sub-routine GDPURDS, some situations require the use of the other subroutines. Once a data set has been completely purged from the data base, it cannot be recovered, because the data tapes have been relabeled as pool tapes. However, if only the catalog entries are changed, the data set can be recovered by rewriting the catalog entries. No tape in the GDB can be used unless its tape label corresponds to the file index record in the file catalog.

In certain cases the GDB manager might want to reference entries in the file and data catalogs without specifying the data name, sequence number, or copy number. This might be necessary if an entry with a "bad" data name or number got into the catalogs by accident. Therefore, certain GDB routines that normally require data name, sequence number, and copy number as input will, instead, use the position of the entry in the catalog. This is specified as follows:

NDNAME = 10H blank

NSEQ = position of the entry in the catalog (including special file catalog or data catalog index record)

The routines that follow this convention are as follows:

GDPFIR – GDB Purge File Index Record

GDPDIR – GDB Purge Data Index Record

GDRFIR – GDB Read File Index Record

GDRDIR – GDB Read Data Index Record

GDRDT, GDRPF, GDRWDI, GDRWFI – These routines follow the same convention but should be used only internally by the GDB system.

To delete a bad GDB data file that cannot be deleted by normal means, the manager writes a GDB main program and calls GDPFIR and GDPDIR as described. If the file is a tape file, the tape can be reentered into the file catalog with GDAPT. If it is a permanent disk data file, the file must be purged by direct use of the system permanent-file control cards.

The manager can recover a data file whose data and file catalog entries are missing by calling subroutine GDSAVE. As long as the tape file itself is intact, the file does not have to be recreated. GDSAVE gets the file index record and data index record off the tape and enters them in the catalogs.

6. SUBROUTINE DOCUMENTATION

6.1 The Index Array

The index array is used to store the index record of the data catalog. The length of the index record as specified by the user will limit the number of words retrieved. Thus, for example, storage does not need to be supplied for the comments unless they are to be used in the program.

INDEX (1) = length of index record in words. Integer.

(2) = data name. R format, seven characters, with the first character nonblank.

(3) = data sequence number. Integer 1–255.

(4) = creation date. Hollerith, as returned by the function ITIMER(1).

(5) = maximum unit record length in words. Integer.

(6) = 0 for fixed-length records.
= 1 for variable-length records.

(7) = number of unit records. Integer.

(8) = number of disk PRUs in index section. Integer.

(9) = number of disk PRUs in address section. Integer.

(10) = number of disk PRUs in data section. Integer.

(11) = number of existence bits. Integer.

(12) = length of comments in words. Integer.

(13)...(n) = existence-bit array.

(n+1)...(m) = comments.

where

n = (INDEX(11)-1)/60 + 13.

m = INDEX(12) + n.

6.2 File Index Record

The array IFNDEX is used to store the file index record.

IFNDEX(1) = file device: tape = 0 or permanent direct access =1.

(2) = file identification. L format. For tape, reel number is given; for permanent file, permanent-file name is given.

(3) = data name. R format.

(4) = data sequence number.

(5) = data copy number.

(6) = creation date. Hollerith, as returned by the function ITIMER(1).

(7) = tape density. Integer.

(8) = length of tape in feet. Integer.

6.3 GDOPEN – GDB Open Catalogs (R)*

GDOPEN attaches the data and file catalogs, which are permanent files residing on the disk.

CALL GDOPEN (FCNAME, FCPASS, POOL, ICOND)

where

FCNAME = permanent-file name for the file catalog. L format.

FCPASS = five-word array containing the permanent-file passwords for the file catalog and the data catalog. The user must specify all the passwords required by subroutines to be called in this run. If the GDB is only to be read, only the read password needs to be given. Permanent-file passwords are in L format, and the order is not significant. If less than five passwords are given, the last one should be followed by a zero word.

POOL = data name of the user's pool tapes. R format.†

ICOND = condition code.

GDOPEN assumes that labeled common block GDLFNS has been set up and initialized in the calling program as follows:

LFNS(1) = logical-file name for data catalog.‡

LFNS(2) = logical-file name for file catalog.

LFNS(3) = logical-file name for mounted input tape.

LFNS(4) = logical-file name for mounted output tape.

LFNS(5) = logical-file name for error-comment output.

LFNS(6) = logical-file name for regular printed output.

---

*The GDB passwords are abbreviated as follows: R = read, E = extend, M = modify, and C = control (see Section 4.4).

†All data names in the GDB are seven characters in R format, the first of which must be nonblank.

‡All logical-file names in the GDB are left-justified with zero file. LFNS(1), LFNS(2), LFNS(5), and LFNS(6) must be mentioned on the PROGRAM card since they are handled with Fortran. All other GDB files are handled outside of Fortran, and therefore must not appear on the PROGRAM card.

6.4 GDADF – GDB Attach Data File (R)

GDADF attaches a data file. If the file resides on the disk as a permanent file, it is attached as a temporary disk file. If it resides on a magnetic tape, the tape is mounted, its GDB label is checked, and it is copied to a temporary disk file. If the data files are to be processed sequentially in the run, it is not necessary (and probably not advisable) to attach them all at the beginning of the run.

CALL GDADF (NDNAME, NSEQ, NCOPY, FNAME, DFPASS, FET, IBUF, LBUF, INDEX, LINDEX, ICOND)

where

NDNAME = data name of file to be attached. R format.

NSEQ = data sequence number:
= 0 means attach highest sequence number.
> 0 means attach NSEQ.

NCOPY = data copy number:
= 0 means attach highest copy number.
> 0 means attach NCOPY.

FNAME = logical-file name of temporary disk file on which the data file is to reside after it is attached or copied to the disk. This file name must not appear on the PROGRAM card. L format.

DFPASS = not used.

FET = eight-word array to hold the file environment table* (FET) for disk file FNAME.

IBUF = buffer (array) used in copying tape to disk. The buffer can be reused after the call to GDADF.

LBUF = length of IBUF. LBUF must be a minimum of 513 words.

INDEX = array containing the data index record for NDNAME NSEQ. This is returned by GDADF. (See Section 6.1.)

LINDEX = number of words to be returned in INDEX.

ICOND = condition code.

*A file environment table is a storage area set up in central memory. It is used by the CDC 6400 operating system and a user program to communicate about the status of a file being read or written.

6.5 GDREAD – GDB Read Data Record (R)

GDREAD returns data records to the calling program, one record at a time. In automatic mode, the user specifies which data records are to be returned by setting up an array IBIT that is similar to the existence-bit portion of the data index record. If a bit in IBIT is set to 1, the corresponding data record will be returned by GDREAD if the record exists. IBIT is not used in the manual mode. The user sets NBIT to one less than the number of the bit in the existence-bit array corresponding to the desired record and calls GDREAD.

```
CALL GDREAD (FNAME, FET, INDEX, MODE, IBIT, NBIT, IBUFF,
             LBUFF, IREC, LREC, MREC, ISTOR, ITYPE, LUERR,
             ICOND)
```

where

FNAME = logical-file name of the file to be read (name with which it was attached in GDADF). L format.

FET = eight-word array set up by GDADF with the FET for disk file FNAME.

INDEX = array containing the data index record for the data file (Section 6.1). The comments section of the index record is not used by GDREAD. The data index record is returned by GDADF.

MODE = 0 means automatic mode.
= 1 means manual mode.

IBIT = array containing bits specifying the data records to be returned. IBIT corresponds to an existence-bit array and is used in the automatic mode only.

NBIT = on input, one less than the bit number in IBIT at which searching is to begin. In the automatic mode, GDREAD increments NBIT until it finds a bit in IBIT that is on and returns the corresponding data record. For the automatic mode, NBIT is initialized to zero (or some other starting point) and is not altered between calls to GDREAD. On output (both modes), NBIT corresponds to the data record returned or not found.

IBUFF = read buffer. Each file being read at the same time by GDREAD must have its own read buffer. The buffer for a given file must not be altered by the user until reading is done on that file. For files with variable-length records, IBUFF is partitioned into two parts: The first is the read buffer for the data themselves, and the second is the read buffer for the address pointers for the file.

LBUFF = buffer sizes:

LBUFF(1) = length of data section of IBUFF, in words.
LBUFF(2) = length of address section of IBUFF. The user must set LBUFF(2) = 0 for fixed-length record data files.
LBUFF(1) and LBUFF (2) must each equal at least 64 words. The larger they are (≤3200 words), the more efficiently the GDB will operate.

IREC = data record returned to calling program. IREC can be any data type.

LREC = number of words of data record to return in IREC.

MREC = actual number of words read.

ISTOR = eight-word scratch array used by GDREAD. Each file being read at the same time by GDREAD must have its own ISTOR array, which must not be altered between calls to GDREAD. ISTOR must be initialized to zero before the first call to read for the file.

ITYPE = read mode for the file (see Section 5.4):

= 1 means forward.
= 2 means reverse.
= 3 means random.

LUERR = logical-file name for error comments = LFNS(5).

ICOND = condition code.

6.6 GDWRIT – GDB Write Data Record

GDWRIT sequentially writes the data records associated with a GDB data file onto a temporary disk file. The user makes repeated calls to GDWRIT, feeding it data records one at a time. GDWRIT must receive data records in the order desired for the output file.

```
CALL GDWRIT (FNAME, FET, INDEX, IBUFF, LBUFF, IREC, LREC,
             ISTOR, LUERR, ICOND)
```

where

FNAME = logical-file name of the temporary disk file on which to write the data. L format.

FET = eight-word array to hold the FET for FNAME.

INDEX = array containing the data index record for the data file (Section 6.1). On the first call to GDWRIT for the file, set:

INDEX(1) = length of INDEX.
INDEX(6) = fixed (=0) or variable (=1) length.

Before the final call to GDWRIT (LREC=0), the user must set all other elements of INDEX except INDEX(8), INDEX(9), and INDEX(10) to the correct values.

IBUFF = write buffer. Each file being written at the same time by GDWRIT must have its own write buffer. The buffer for a given file must not be altered by the user until writing is done on that file. For files with variable-length records, IBUFF is partitioned into two parts; the first is the write buffer for the data themselves, and the second, the write buffer for the address pointers for the file.

LBUFF = buffer sizes:

LBUFF(1) = length of data section of IBUFF.
LBUFF(2) = length of address section of IBUFF. The user must set LBUFF(2) = 0 for fixed-length record data files.
LBUFF(1) and LBUFF(2) must each equal at least 64 words. The larger they are (≤3200 words), the more efficiently the GDB will operate.

IREC = record to be written. IREC can be any data type.

LREC = length of IREC. After the last call to GDWRIT to write a data record, it is necessary to make a final call with LREC = 0 in order to flush the write buffers and perform other final operations on the file.

ISTOR = eight-word scratch array used by GDWRIT. Each file being written at the same time must have its own ISTOR array, which must not be altered between calls to GDWRIT. ISTOR(1) must be initialized to zero and ISTOR(4) to the maximum number of records to be written for this file. Setting ISTOR(4) is necessary only when dealing with variable-length records.

LUERR = logical-file name for error comments = LFNS(5).

ICOND = condition code.

6.7 GDREWR – GDB Rewrite Data Record

GDREWR writes the data records associated with a GDB data file onto a temporary disk file. Unlike GDWRIT, however, it does not require the records to be input sequentially. Furthermore, GDREWR allows the user to rewrite selected data records on an existing data file (not a permanent file) without copying the entire file.

In writing a new file, GDREWR is called once for each data record in the file. The records can come in any order since direct-access methods are used and the calling program gives GDREWR the relative position of the data record in the file.

To make changes to an existing GDB file, the user need only attach the file with GDADF and then call GDREWR for each record to be changed. For fixed-length records, the new version of the record must be the same length as the old one, and for variable-length records, the new version can be any length.

GDREWR will operate most efficiently if the incoming data records are at least partially ordered. If the records skip around with no order, GDREWR could have to do one random-access read and one random-access write for each data record.

```
CALL GDREWR (FNAME, FET, INDEX, IBUFF, LBUFF, IREC, LREC,
             ISTOR, NBIT, ITYPE, LUERR, ICOND)
```

where

FNAME = logical-file name of the temporary disk file on which to write the data. If the file already exists and is to be altered, FNAME is the logical-file name with which it was attached in GDADF. L format.

FET = eight-word array to hoid the FET for FNAME.

INDEX = array containing the data index record for the tile (Section 6.1). INDEX must be completely set up before the first call to GDREWR (except that if FNAME is a new file, then INDEX(8), INDEX(9), and INDEX(10) are set up by GDREWR). This means that the existence-bit array must be set up before the first call to GDREWR.

IBUFF = read/write buffer. Each file being written at the same time must have its own buffer and it must not be altered by the user until writing has been completed on that file. For files with variable-length records, IBUFF is partitioned into two parts: The first part is the buffer for the data themselves, and the second, the buffer for the address pointers for the file.

LBUFF = buffer sizes:

LBUFF(1) = length of data section of IBUFF.
LBUFF(2) = length of address section of IBUFF. The user must set LBUFF(2) = 0 for fixed-length record data files.
LBUFF(1) and LBUFF(2) must each equal at least 64 words. In general, the larger they are (≤3200 words), the more efficiently the GDB will operate.

IREC = record to be written.

LREC = length of IREC. After the last call to GDREWR to write a record, it is necessary to make a final call to GDREWR with LREC = 0 in order to flush the buffers and perform other final operations on the file (i.e., INDEX(10) is recomputed and INDEX is rewritten at the start of the file).

ISTOR = eight-word scratch array used by GDREWR. Each file being written at the same time must have its own ISTOR array, which must not be altered between calls to GDREWR. ISTOR must be initialized to zero before the first call to GDREWR.

NBIT = existence-bit number of record to be written.

ITYPE = signal telling current order of data records (write mode):

= 1 means forward.
= 2 means reverse.
= 3 means random.

This signal should be changed during the writing of the file as the order of the data records dictates (see Section 5.4). If the data records are not ordered in any particular way, set ITYPE = 3.

LUERR = logical unit for error comments = LFNS(5).

ICOND = condition code.

6.8 GDRDIR – GDB Read Data Index Record (R)

GDRDIR reads the data index record for a data file from the data catalog.

```
CALL GDRDIR (NDNAME, NSEQ, INDEX, LINDEX, ICOND)
```

where

NDNAME = data name. R format.

NSEQ = data sequence number:
- = 0 means highest sequence number.
- > 0 means NSEQ.

INDEX = array containing the data index record (Section 6.1).

LINDEX = number of words of data index record to return in INDEX.

ICOND = condition code.

6.9 GDRFIR – GDB Read File Index Record (R)

GDRFIR reads the file index record for a GDB file from the file catalog.

```
CALL GDRFIR (NDNAME, NSEQ, NCOPY, IFNDEX, LFNDEX, ICOND)
```

where

NDNAME = data name. R format.

NSEQ = data sequence number:
- = 0 means highest sequence number.
- > 0 means NSEQ.

NCOPY = data copy number:
- = 0 means highest copy number.
- > 0 means NCOPY.

IFNDEX = array containing the file index record.

LFNDEX = length of IFNDEX.

ICOND = condition code.

6.10 GDPCAT – GDB Print Catalogs (R)

GDPCAT prints the data or file catalog. The catalog can be optionally sorted before printing.

```
CALL GDPCAT (ICAT, ISORT, NCOPY, IPRINT, INDEX, LINDEX, ICOND)
```

where

ICAT	=	1 means file catalog. = 2 means print data catalog.
ISORT	=	0 means no sort first. = 1 means sort before printing.
NCOPY	=	number of print copies desired (not implemented).
IPRINT	=	applies to data catalog only: = 0 means minimum printout. = 1 means limited (minimum plus comments). = 2 means full (minimum plus comments plus existence bits).
INDEX	=	scratch array used by GDPCAT to hold the file index records or data index records to be printed.
LINDEX	=	length of INDEX. Size of a file index record or largest data index record.
ICOND	=	condition code.

6.11 GDRCAT – GDB Release Catalogs (R)

GDRCAT allows the user to return one version of the GDB catalogs so that a new one can be attached by calling GDOPEN again.

```
CALL GDRCAT
```

There are no arguments in the calling sequence.

6.12 GDRDISK – GDB Return Disk File (R)

GDRDISK allows the user to return the disk file on which a data file resides when work on it is completed so that another data file can then be attached with the same logical-file name and file environment table.

```
CALL GDRDISK (FNAME, FET)
```

where

FNAME	=	logical-file name of the temporary disk file to be returned.
FET	=	eight-word array containing the file environment table associated with FNAME.

6.13 GDCDF – GDB Create Data File (R, E)

GDCDF makes the temporary file written by GDWRIT a permanent part of the GDB. It is cataloged as a permanent disk file or copied to tape. GDCDF must be followed by a call to GDEUP.

```
CALL GDCDF (FNAME, FET, IBUF, LBUF, INDEX, NPERM, DFPASS,
            PFID, NCOPY, ICOND)
```

where

FNAME = logical-file name of temporary disk file on which the data file resides. L format.

FET = eight-word array containing the FET for FNAME.

IBUF = buffer (array) used in copying disk to tape. The buffer can be reused after the call to GDCDF.

LBUF = length of IBUF. LBUF must be a minimum of 513 words.

INDEX = array containing the data index record for the data file (see Section 6.1).

NPERM = "name" means catalog FNAME as permanent-file "name." L format.
= 0 means no permanent-file copy.

DFPASS = not used.

PFID = permanent-file identification of the user, which is necessary to catalog a new permanent file (NPERM ≠ 0). L format.

NCOPY = number of tape copies of FNAME to be made.

ICOND = condition code.

6.14 GDEUP – GDB End Update (R, E)

GDEUP closes (updates) the data and file catalogs and creates backups for the new versions. The last operation performed must be a call to GDEUP in any run that changes the GDB.

CALL GDEUP (M1, M2, N1, N2, INDEX, LINDEX, ICOND)

where

M1 = number of backups of file catalog to be made.

M2 = number of backups of data catalog to be made.

N1 = print signal for file catalog:
- = 0 means minimum.
- = 1 means limited.
- = 2 means full.

N2 = print signal for data catalog:
- = 0 means minimum printout.
- = 1 means limited (minimum plus comments).
- = 2 means full print (minimum plus comments plus existence bits).

INDEX = array large enough to hold the largest data index record in the data catalog (see Section 6.1).

LINDEX = length of INDEX.

ICOND = condition code.

6.15 GDAPT – GDB Add Pool Tape (R, E)

GDAPT adds a magnetic tape to the GDB as a pool tape by adding it to the file catalog and writing a GDB tape label on the tape.

```
CALL GDAPT (NTAPE, IDENS, LEN, ICOND)
```

where

NTAPE	=	visual reel number of the tape. L format.
IDENS	=	density at which the tape is to be read and written. The value is an integer, usually 556 bpi.
LEN	=	length of the tape in feet. Integer.
ICOND	=	condition code.

Currently, IDENS and LEN are not used by the GDB but are retained in the file catalog.

6.16 GDDEACT – GDB Deactivate Pool Tape (R, E)

GDDEACT deactivates a pool tape so that it will not be requested in future GDB runs.

```
CALL GDDEACT (NDNAME, NSEQ, NCOPY, IFNDEX, LFNDEX, ICOND)
```

where

NDNAME	=	data name. R format.
NSEQ	=	data sequence number.
NCOPY	=	data copy number.
IFNDEX	=	array containing the file index record written in the file catalog. This is returned by GDDEACT.
LFNDEX	=	length of IFNDEX.
ICOND	=	condition code.

GDDEACT changes the data name of a bad pool tape to BADPOOL. The GDB manager should periodically delete BADPOOL entries from the file catalog.

Note that GDDEACT requires only read and extend permission.

6.17 GDWDIR – GDB Write Data Index Record (R, E)

GDWDIR adds a data index record to the data catalog, first checking to see that the data name and sequence number in INDEX are not duplicates.

```
CALL GDWDIR (INDEX, ICOND)
```

where

INDEX = array containing the data index record to be added to the data catalog (see Section 6.1).

ICOND = condition code.

6.18 GDAFIR – GDB Add File Index Record (R, E)

GDAFIR adds a file index record to the file catalog, first making sure that the file index record does not duplicate an existing data name, sequence number, or copy number.

```
CALL GDAFIR (IFNDEX, LFNDEX, ICOND)
```

where

IFNDEX = array containing the file index record to be added.

LFNDEX = length of IFNDEX.

ICOND = condition code.

6.19 GDPURDS – GDB Purge Data Set (R, E, M)

GDPURDS, the basic file-removal routine for the GDB, removes an entire data set from the GDB system.

```
CALL GDPURDS (NDNAME, NSEQ, PASS, ICOND)
```

where

NDNAME = data name. R format.

NSEQ = data sequence number:
- = 0 means remove all NDNAME entries except that with the highest sequence number.
- = -1 means remove all NDNAME entries.

PASS = not used.

ICOND = condition code.

GDPURDS takes care of the entire purging operation by: 1) removing data index record(s) from the data catalog, 2) reassigning data tapes as POOL tapes, 3) purging permanent-file versions of the data file from the disk, and 4) making appropriate changes to the file catalog.

6.20 GDRDT – GDB Release Data Tape (R, E, M)

GDRDT sets data tapes back to pool-tape status by altering the file catalog and rewriting the GDB label on the tape.

```
CALL GDRDT (NDNAME, NSEQ, NCOPY, ICOND)
```

where

NDNAME = data name. R format.

NSEQ = data sequence number:
- = 0 means remove all NDNAME entries except that with the highest sequence number.
- = -1 means remove all NDNAME entries.

NCOPY = data copy number:
- = 0 means remove all entries with data name NDNAME and sequence number NSEQ except that copy with the highest copy number.
- = -1 means remove all entries with data name NDNAME and sequence number NSEQ.

ICOND = condition code.

GDRDT does not alter the data catalog.

6.21 GDRPF – GDB Release Permanent File (R, E, M)

GDRPF releases the permanent-file version of a GDB data file by removing its file index record from the file catalog and purging the permanent file. This does not alter the data catalog.

CALL GDRPF (NDNAME, NSEQ, NCOPY, PASS, ICOND)

where

NDNAME = data name. R format.

NSEQ = data sequence number:

- = 0 means remove all NDNAME entries except that with the highest sequence number.
- = -1 means remove all NDNAME entries.

NCOPY = data copy number:

- = 0 means remove all entries with data name NDNAME and sequence number NSEQ except that copy with the highest copy number.
- = -1 means remove all entries with data name NDNAME and sequence number NSEQ.

PASS = not used.

ICOND = condition code.

6.22 GDPDIR – GDB Purge Data Index Record (R, E, M)

GDPDIR purges an entry from the data catalog by assigning it data name DELETE.

```
CALL GDPDIR (NDNAME, NSEQ, ICOND)
```

where

NDNAME = data name. R format.

NSEQ = data sequence number:
- = 0 means remove all NDNAME entries except that with the highest sequence number.
- = -1 means remove all NDNAME entries.

ICOND = condition code.

6.23 GDPFIR – GDB Purge File Index Record (R, E, M)

GDPFIR purges an entry from the file catalog by assigning it data name DELETE.

```
CALL GDPFIR (NDNAME, NSEQ, NCOPY, ICOND)
```

where

NDNAME = data name. R format.

NSEQ = data sequence number:
- = 0 means remove all NDNAME entries except that with the highest sequence number.
- = -1 means remove all NDNAME entries.

NCOPY = data copy number:
- = 0 means remove all entries with data name NDNAME and sequence number NSEQ except that copy with the highest copy number.
- = -1 means remove all entries with data name NDNAME and sequence number NSEQ.

ICOND = condition code.

6.24 GDRWDI – GDB Rewrite Data Index Record (R, E, M)

GDRWDI overwrites a data index record specified by NDNAME and NSEQ with INDEX.

```
CALL GDRWDI (NDNAME, NSEQ, INDEX, ICOND)
```

where

NDNAME = data name. R format.

NSEQ = data sequence number:
- = 0 means remove all NDNAME entries except that with the highest sequence number.
- = -1 means remove all NDNAME entries.

INDEX = array containing the data index record to be written in place of the old one (see Section 6.1).

ICOND = condition code.

6.25 GDRWFI – GDB Rewrite File Index Record (R, E, M)

GDRWFI overwrites the file index record for a data file specified by NDNAME, NSEQ, and NCOPY with the contents of IFNDEX.

CALL GDRWFI (NDNAME, NSEQ, NCOPY, IFNDEX, LFNDEX, ICOND)

where

NDNAME = data name. R format.

NSEQ = data sequence number:
- = 0 means remove all NDNAME entries except that with the highest sequence number.
- = -1 means remove all NDNAME entries.

NCOPY = data copy number:
- = 0 means remove all entries with data name NDNAME and sequence number NSEQ except that copy with the highest copy number.
- = -1 means remove all entries with data name NDNAME and sequence number NSEQ.

IFNDEX = array containing the new file index record (see Section 6.2). The contents of IFNDEX replace the old file index record.

LFNDEX = length of IFNDEX.

ICOND = condition code.

6.26 GDSAVE – GDB Save a Data Tape (R, E, M)

GDSAVE allows the GDB manager to recover a data tape file if its data catalog entry and file catalog entry are missing but the tape file itself is intact. This might occur if the catalogs are reloaded from backups that were created before the data file was created. It might also occur if the program that created the data file failed after the file was created by GDCDF but before GDEUP was called. GDSAVE requests the tape on which the data file resides and copies the file index record and data index record from the tape into the file catalog and data catalog, respectively. Only one tape copy in a data set can be recovered in this manner.

CALL GDSAVE (NVIS, IFNDEX, LFNDEX, INDEX, LINDEX, ICOND)

where

NVIS	=	visual reel number of the tape on which the data file resides. A format.
IFNDEX	=	buffer for file index record.
LFNDEX	=	length of IFNDEX.
INDEX	=	buffer large enough to hold the entire data index record.
LINDEX	=	length of INDEX.
ICOND	=	condition code.

6.27 GDRELOD – GDB Reload Catalog (R, E, C)

GDRELOD reloads the data or file catalog from a backup tape specified by the user. The tape is copied to the disk, the old version of the catalog is purged (if it exists), and the new version is cataloged as a permanent file. GDRELOD might be called for two reasons: The data or file catalogs may be too large or their permanent files may have been destroyed owing to system failure. Repeated use of the GDB causes the permanent files to get cluttered with inactive entries and to keep growing in number of disk PRUs occupied, even though their apparent sizes have not increased very much. Dumping the catalogs to tape and then reloading enable all the inactive entries to be deleted. This process should be done periodically so the user's permanent-file allotment is not exceeded. To dump the catalogs to tape in a normal run of the GDB, the user calls GDEUP with M1 and M2 (number of backups) equal to 1. Then, in a separate run of the GDB, GDRELOD is called. If both catalogs are to be reloaded, GDRELOD can be called twice in one run.* Reloading the catalogs must be performed in a separate, special job in which GDOPEN and GDEUP are not called.

```
CALL GDRELOD (NDNAME, PFNAME, PFID, PASS, NVIS, INDEX,
              LINDEX, ICOND)
```

where

NDNAME = 7RFILECAT or 7RDATACAT.

PFNAME = permanent-file name of the catalog to be reloaded. L format.

PFID = permanent-file identification of the user (needed to catalog a new permanent file).

PASS = five-word array containing the permanent-file passwords needed to attach and purge the old version of NDNAME and then to catalog the new version. L format.

PASS(1) = turnkey, set to zero if password is not to be specified.
PASS(2) = control, set to zero if password is not to be specified.
PASS(3) = modify, set to zero if password is not to be specified.
PASS(4) = extend, set to zero if password is not to be specified.
PASS(5) = read, set to zero if password is not to be specified.

*Note that the sequence numbers of the data and file catalogs must always match.

NVIS = visual reel number of the tape on which the backup resides. A format.

INDEX = array large enough to hold largest data index record or file index record (see Section 6.1).

LINDEX = length of INDEX.

ICOND = condition code.

6.28 GDCREAT – GDB Create Initial Data and File Catalog (R, E, M, C)

GDCREAT sets up an initial data and file catalog for a new GDB. Once this is done, any other GDB subroutines can be used as usual in subsequent runs.

GDCREAT (NAMEFC, NAMEDC, PFID, PASS, ICOND)

where

NAMEFC = permanent-file name for file catalog. L format.

NAMEDC = permanent-file name for data catalog. L format.

PFID = identification for user's permanent-file allotment. L format.

PASS = five-word array containing the permanent-file passwords for the catalogs. L format.

PASS(1) = turnkey = 0.
PASS(2) = control.
PASS(3) = modify.
PASS(4) = extend.
PASS(5) = read.

ICOND = condition code.

GDCREAT is called in the same way as any other GDB subroutine, except that GDOPEN and GDEUP must not be called.

6.29 EXBIT

EXBIT sets the existence bit corresponding to LAT and LONG in an array.

CALL EXBIT (IDEX, NEX, LAT, LONG, NBIT, ISIG)

where

IDEX	=	array in which bit is to be set.
NEX	=	number of bits in IDEX.
LAT	=	latitude (degrees).
LONG	=	longitude (degrees).
NBIT	=	existence-bit number corresponding to LAT, LONG. Output.
ISIG	=	0 means normal exit. 1 means bad exit.

LAT and LONG are the latitude and longitude of the northwest corner of the 1°×1° square corresponding to NBIT. The order of these squares is assumed to be longitude 1° to 360° within latitude 90° to -89°.

6.30 LATLON

LATLON gets the latitude and longitude from the existence-bit number.

```
CALL LATLON (NBIT, LAT, LONG)
```

where

NBIT = existence-bit number.

LAT = latitude (degrees).

LONG = longitude (degrees).

LAT and LONG are the latitude and longitude of the 1°×1° square corresponding to NBIT. The order of these squares is assumed to be longitude 1° to 360° within latitude 90° to -89°.

6.31 IBIT

IBIT is a function that sets a bit in an array to 0 or 1.

```
ISIG = IBIT (ITEM, IWRDS, NBITS, I1)
```

where

ISIG	=	0 means normal exit. = 1 means bad exit.
ITEM	=	bit number.
IWRDS	=	array.
NBITS	=	number of bits in IWRDS.
I1	=	value to which bit number "item" is to be set (= 0 or 1).

6.32 NETBIT

NETBIT is a function that gets a bit from an array.

IVAL = NETBIT (ITEM, IWRDS, NBITS, ICODE)

where

IVAL	=	value of bit retrieved (= 0 or 1).
ITEM	=	bit to be retrieved.
IWRDS	=	array.
NBITS	=	number of bits in IWRDS.
ICODE	=	0 means normal exit. = 1 means bad exit.

6.33 Error Codes

The variable ICOND is an error flag that appears in the calling sequence of most GDB subroutines. If an error is encountered, the subroutine prints an error comment and returns control to the calling program. The user should test ICOND after each call to a GDB subroutine. The following table lists ICOND values.

ICOND =

0 = O.K. No error.

1 = general bad exit code.

2 = call to attach a permanent file failed.

3 = bad data name, sequence number, or copy number.

4 = file catalog already full.

5 = data catalog already full.

6 = cannot find (data name, sequence number, copy number) in file catalog.

7 = cannot find (data name, sequence number) in data catalog.

8 = duplicate (data name, sequence number) in data catalog.

9 = trouble packing or unpacking identification word.

10 = duplicate (data name, sequence number, copy number) in file catalog.

11 = trouble requesting a tape (system routine REQUEST).

12 = visual reel number or data name on tape label does not match file catalog; wrong tape may be mounted.

13 = data catalog or file catalog does not have copy number equal to 1.

14 = new record is longer than old record, so random-access rewrite is illegal.

15 = call to extend a permanent file failed.

16 = call to open a random-access disk file with OPENRA failed.

17 = not used.

18 = call to catalog a permanent file failed.

19 = buffer to copy to or from tape is too small (< 513 words).

20 = call to RDBUF or WRBUF failed to read or write a tape.

21 = call to PRIMRA package failed.

22 = user wants a data point that does not exist in the data file.

23 = attempt to write more data records than specified maximum (variable-length records only).

24 = call to purge a permanent file failed.

25 = attempt to read past end of information on a random file.

26 = nonzero code and status in call to CIOC.

27 = buffer too small to hold the data or file index record.

28 = attempt to rewrite in place on a permanent file.

29 = illegal attempt to purge file or data index record; no modify permission.

30 = attempt to deactivate a nonpool tape.

31 = duplicate visual reel number in file catalog.

32 = no modify permission.

7. SAMPLE USER PROGRAMS (code only)

```
      PROGRAM TEST01(INPUT,OUTPUT,TAPE1,TAPE2,TAPE99=OUTPUT)
C         WRITE A NEW GDB DATA FILE WITH FIXED LENGTH RECORDS
C         SEQUENTIAL WRITE
C         ONE TAPE COPY ONLY
C         BACKUP DATA AND FILE CATALOGS
C
      COMMON/GDLFNS/LFNS(6)
      DIMENSION FCPASS(5)
      DATA FCPASS/7LREDWOOD,3LELM,3*0/
      DATA FCNAME,POOL/6LGDFILE,7RGDBPOOL/
      DIMENSION IFNDEX(8),INDEX(1500)
      DATA LFNDEX,LINDEX,LIND/8,1500,99/
      INTEGER FET(8)
      DATA FNAME/6LTAPE60/
      DATA LFNS/5LTAPE1,5LTAPE2,5LTAPE3,5LTAPE4,6LTAPE99,6LTAPE99/
      DATA MZERO/-0/
      DIMENSION IBUFF(3200),LBUFF(2)
      DATA LBUFF/3200,0/
      DIMENSION ISTOR(8)
      DATA ISTOR/8*0/
      DATA NPERM,MCOPY/0,1/
      DIMENSION IREC(3)
      DATA LREC/3/
      LUERR=LFNS(5)


C         OPEN DATA AND FILE CATS.
      CALL GDOPEN(FCNAME,FCPASS,POOL,ICOND)
      IF(ICOND.GT.0) GO TO 900
C
C         CREATE DATA INDEX RECORD
      INDEX(1)=LIND
      INDEX(2)=7RSAMPLE1
      INDEX(3)=1
      INDEX(4)=ITIMER(1)
      INDEX(5)=3
      INDEX(6)=0
      INDEX(7)=5000
      INDEX(8)=0
      INDEX(9)=0
      INDEX(10)=0
      INDEX(11)=5000
      INDEX(12)=3
      DO 100 I=1,84
      INDEX(I+12)=MZERO
  100 CONTINUE
      INDEX(97)=10HEXISTENCE
      INDEX(98)=10HBITS ARE A
      INDEX(99)=10HLL ON
C
C
C         WRITE DATA FILE
      DO 200 I=1,5000
      IREC(1)=0
      IREC(2)=0
      IREC(3)=I
      CALL GDWRIT(FNAME,FET,INDEX,IBUFF,LBUFF,
     1IREC,LREC,ISTOR,LUERR,ICOND)
      IF(ICOND.GT.0) GO TO 900
  200 CONTINUE
      CALL GDWRIT(FNAME,FET,INDEX,IBUFF,LBUFF,
     1IREC,0,ISTOR,LUERR,ICOND)



      IF(ICOND.GT.0) GO TO 900
C
C
C         CREATE GDB DATA FILE
      CALL GDCDF(FNAME,FET,IBUFF,LBUFF,INDEX,NPERM,DFPASS,PFID,
     1MCOPY,ICOND)
      IF(ICOND.GT.0) GO TO 900
C
C         CLOSE DATA AND FILE CATALOGS
C         BACKUP DATA AND FILE CATALOGS
      CALL GDEUP(1,1,1,1,INDEX,LINDEX,ICOND)
      STOP
  900 CONTINUE
      CALL ABNORML
      END
```

```
      PROGRAM TEST02(INPUT,OUTPUT,TAPE1,TAPE2,TAPE99=OUTPUT)
C         READ AND PRINT TWO GDB DATA FILES
C         AUTOMATIC MODE
C         FORWARD READ
C         FIXED LENGTH UNIT RECORDS
C
      COMMON/GDLFNS/LFNS(6)
      DIMENSION FCPASS(5)
      DATA FCPASS/7LREDWOOD,4*0/
      DATA FCNAME,POOL/6LGDFILE,7RGDBPOOL/
      DIMENSION IFNDEX(8),INDEX(1500)
      DATA LFNDEX,LINDEX/8,1500/
      DATA LFNS/5LTAPE1,5LTAPE2,5LTAPE3,5LTAPE4,6LTAPE99,6LTAPE99/
      INTEGER FET(8)
      DATA FNAME/6LTAPE60/
      DIMENSION IBUFF(3200)
      DIMENSION LBUFF(2)
      DATA LBUFF/3200,0/
      DIMENSION ISTOR(8)
      DATA ISTOR/8*0/
      DIMENSION IREC(3)
      DATA LREC/3/
      DIMENSION IBIT(1500)
      EQUIVALENCE(IBIT,INDEX(13))
      DIMENSION NDNAME(2),NSEQ(2),NCOPY(2)
      DATA NDNAME/7RSAMPLE1,7RSAMPLE2/
      DATA NSEQ/1,1/
      DATA NCOPY/0,0/
      DATA MODE,NBIT,ITYPE/0,0,1/
      LUERR=LFNS(5)

C         OPEN DATA AND FILE CATS.
      CALL GDOPEN(FCNAME,FCPASS,POOL,ICOND)
      IF(ICOND.GT.0) GO TO 900
C
C         ATTACH DATA FILE (AND GET DATA INDEX RECORD)
      DO 300 J=1,2
      CALL GDADF(NDNAME(J),NSEQ(J),NCOPY(J),FNAME,DFPASS,FET,
     1IBUFF,LBUFF,INDEX,LINDEX,ICOND)
      IF(ICOND.GT.0) GO TO 900

C         READ AND PRINT ALL EXISTING UNIT RECORDS IN THE FILE
      NRECS=INDEX(7)
      NBIT=0
      DO 220 JJ=1,8
  220 ISTOR(JJ)=0
      DO 200 I=1,NRECS
      IREC(1)=0
      IREC(2)=0
      IREC(3)=0
      CALL GDREAD(FNAME,FET,INDEX,MODE,IBIT,NBIT,IBUFF,LBUFF,
     1IREC,LREC,MREC,ISTOR,ITYPE,LUERR,ICOND)
      IF(ICOND.GT.0) GO TO 900

      PRINT 210,NBIT,MREC,IREC
  210 FORMAT(5I10)
  200 CONTINUE

C         RELEASE DISK FILE FNAME SO IT CAN BE REUSED
      CALL GDRDISK(FNAME,FET)



  300 CONTINUE
      STOP
  900 CONTINUE
      CALL ABNORML
      END
```

```
      PROGRAM TEST03(INPUT,OUTPUT,TAPE1,TAPE2,TAPE99=OUTPUT)
C         GDB TEST - CREATE GDB DATA FILE WITH VARIABLE LENGTH RECS.
C         RANDOM WRITE
C         REVERSE ORDER
C         ONE TAPE COPY
C
C
      COMMON/GDLFNS/LFNS(6)
      DIMENSION IFNDEX(8),INDEX(1500)
      DATA LFNDEX,LINDEX,LIND/8,1500,17/
      DATA LFNS/5LTAPE1,5LTAPE2,5LTAPE3,5LTAPE4,6LTAPE99,6LTAPE99/
      DIMENSION FCPASS(5)
      DATA FCPASS/7LREDWOOD,3LELM,3*0/
      DATA FCNAME,POOL/6LGDFILE,7RGDBPOOL/
      INTEGER FET(8)
      DATA FNAME/6LTAPE60/
      DIMENSION IBUFF(1024),LBUFF(2)
      DATA LBUFF/512,512/
      DIMENSION ISTOR(8)
      DATA ISTOR/8*0/
      DATA NPERM/0/
      DIMENSION IREC(100)
C
      LBTOT=LBUFF(1)+LBUFF(2)
      LUERR=LFNS(5)
      MCOPY=1
      MZERO=777777777777777777777B
C
C         OPEN CATALOGS
      CALL GDOPEN(FCNAME,FCPASS,POOL,ICOND)
      IF(ICOND.GT.0) GO TO 900
C
C         SET UP DATA INDEX RECORD
      INDEX(1)=LIND
      INDEX(2)=7RVAR0001
      INDEX(3)=1
      INDEX(4)=ITIMER(1)
      INDEX(5)=100
      INDEX(6)=1
      INDEX(7)=100
      INDEX(8)=0
      INDEX(9)=0
      INDEX(10)=0
      INDEX(11)=100
      INDEX(12)=3
      INDEX(13)=MZERO
      INDEX(14)=MZERO
      INDEX(15)=10HTHIS IS A
      INDEX(16)=10HTEST OF VA
      INDEX(17)=10HRIABLE LEN
C
C         WRITE UNIT RECORDS IN REVERSE ORDER
C
      LAST=INDEX(7)
      ITYPE=2
      DO 110 II=1,LAST
      NBIT=LAST-II+1
      LREC=NBIT
      DO 50 J=1,LREC
      IREC(J)=0
   50 CONTINUE
      IREC(LREC)=LREC



      CALL GDREWR(FNAME,FET,INDEX,IBUFF,LBUFF,IREC,LREC,
     1ISTOR,NBIT,ITYPE,LUERR,ICOND)
      IF(ICOND.GT.0) GO TO 900
  110 CONTINUE
      CALL GDREWR(FNAME,FET,INDEX,IBUFF,LBUFF,IREC,0,
     1ISTOR,NBIT,ITYPE,LUERR,ICOND)
      IF(ICOND.GT.0) GO TO 900
C
C         CREATE GDB DATA FILE
C
      CALL GDCDF(FNAME,FET,IBUFF,LBTOT,INDEX,NPERM,DFPASS,PFID,
     1MCOPY,ICOND)
      IF(ICOND.GT.0) GO TO 900
C
C
      CALL GDEUP(0,0,1,1,INDEX,LINDEX,ICOND)
      STOP
  900 CONTINUE
      CALL ABNORML
      END
```

```
      PROGRAM TEST04(INPUT,OUTPUT,TAPE1,TAPE2,TAPE99=OUTPUT)
C         READ AND PRINT GDB DATA FILE
C         VARIABLE LENGTH UNIT RECORDS
C         READ IN REVERSE ORDER (MANUAL MODE)
C
      COMMON/GDLFNS/LFNS(6)
      DIMENSION FCPASS(5)
      DATA FCPASS/7LREDWOOD,4*0/
      DATA FCNAME,POOL/6LGDFILE,7RGDBPOOL/
      DIMENSION IFNDEX(8),INDEX(1500)
      DATA LFNDEX,LINDEX/8,1500/
      DATA LFNS/5LTAPE1,5LTAPE2,5LTAPE3,5LTAPE4,6LTAPE99,6LTAPE99/
      INTEGER FET(8)
      DATA FNAME/6LTAPE60/
      DIMENSION IBUFF(1024),LBUFF(2)
      DATA LBUFF/512,512/
      DIMENSION ISTOR(8)
      DATA ISTOR/8*0/
      DIMENSION IREC(100)
      DATA LREC/100/
      LBTOT=LBUFF(1)+LBUFF(2)
      LUERR=LFNS(5)


C         OPEN DATA AND FILE CATS.
      CALL GDOPEN(FCNAME,FCPASS,POOL,ICOND)
      IF(ICOND.GT.0) GO TO 900
C

C         ATTACH DATA FILE (AND GET DATA INDEX RECORD)
      NDNAME=7RVAR0001
      NSEQ=1
      NCOPY=1
      CALL GDADF(NDNAME,NSEQ,NCOPY,FNAME,DFPASS,FET,IBUFF,LBTOT,INDEX,
     1LINDEX,ICOND)
      IF(ICOND.GT.0) GO TO 900

C         READ AND PRINT FILE IN REVERSE ORDER
C         MANUAL MODE- IBIT IS NOT USED
      MODE=1
      ITYPE=2
      NRECS=INDEX(7)
      DO 200 I=1,NRECS
C  FOR READ, SET NBIT TO ONE LESS THAN THE DESIRED UNIT RECORD
      NBIT=100-I
      CALL GDREAD(FNAME,FET,INDEX,MODE,IBIT,NBIT,IBUFF,LBUFF,
     1IREC,LREC,MREC,ISTOR,ITYPE,LUERR,ICOND)
      IF(ICOND.EQ.22) GO TO 200
      IF(ICOND.GT.0) GO TO 900
      PRINT 210,MREC,IREC(MREC)
  210 FORMAT(2I5)
  200 CONTINUE
      STOP
  900 CONTINUE
      CALL ABNORML
      END
```

APPENDIX A

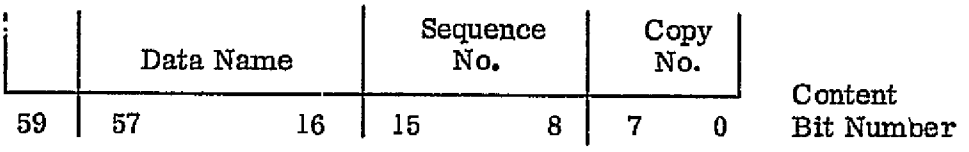
STRUCTURE DIAGRAMS OF THE GDB SYSTEM ORGANIZATION

APPENDIX A

STRUCTURE DIAGRAMS OF THE GDB SYSTEM ORGANIZATION

The following block diagrams describe the organization of the GDB subroutines. Some of these were not mentioned in the previous documentation since they are internal to the GDB system and are never called by the user. Note that many GDB subroutines that the user may call are also called by other subroutines.

Internally to the GDB, each entry in the data catalog or file catalog is identified by a unique ID word. Subroutines DNAM, MAKEID, GETID, and LOOKUP manipulate this ID word, which contains the following information packed into one 60-bit computer word:



Most of the names given in the diagrams are individual subroutine or function names. However, some represent a package of routines. These are described below.

FORTRAN random-access routines:

OPENMS – open file

READMS – random read

WRITMS – random write

RDBUF package, which performs regular sequential input/output, avoiding Fortran completely:

OPBUF – open file

RDBUF – read

WRBUF – write

RWBUF – rewind

EOFBUF – write end of file

UNLOD – unload

PRIMRA package, which performs random-access input/output, avoiding Fortran:

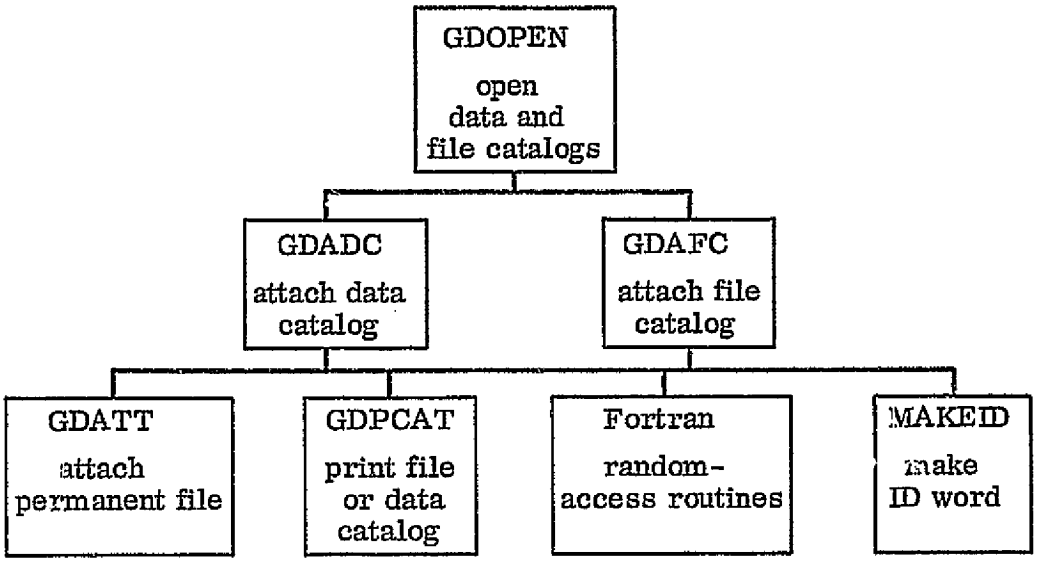
OPENRA – open the file

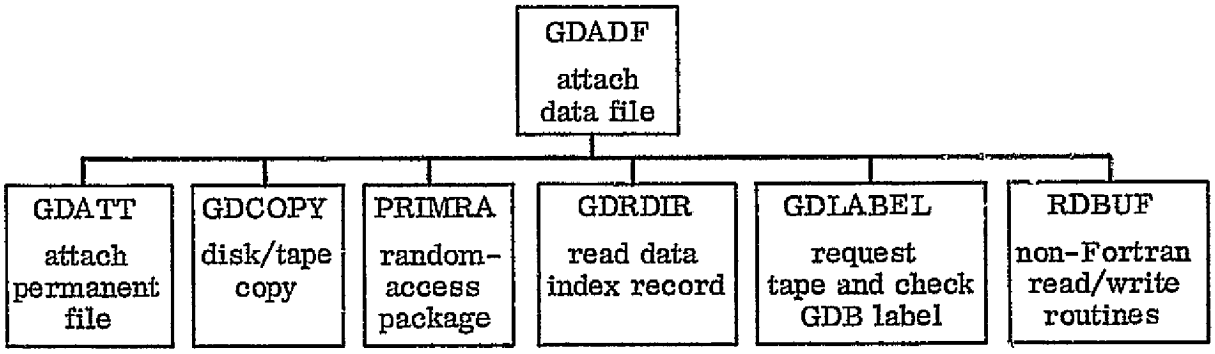
READRA – random read

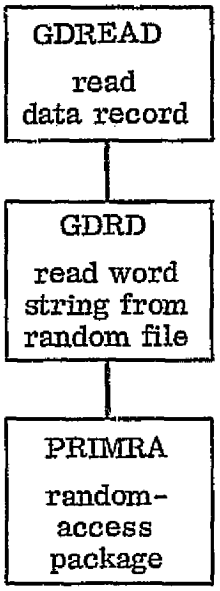
WRITRA – random write

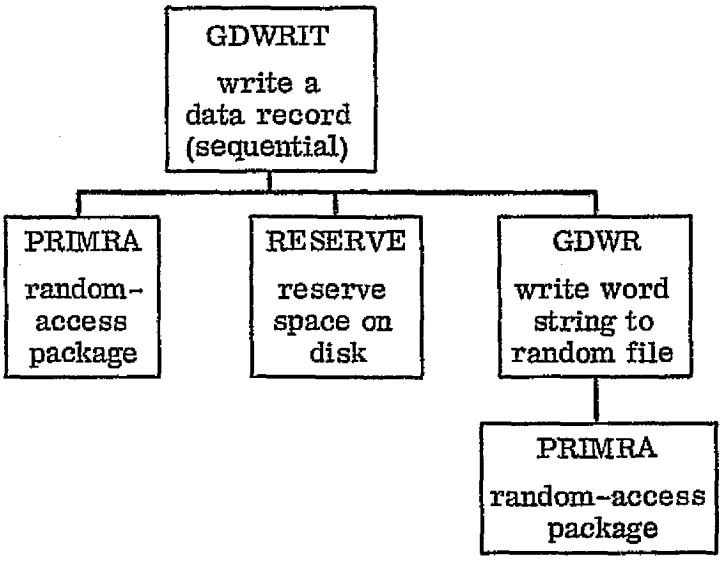
REWRRA – random rewrite in place

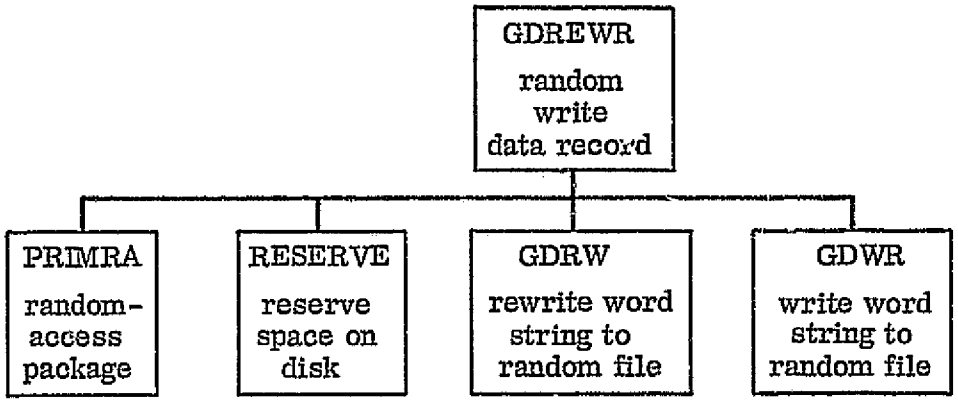
Subroutine EPRINT is called by nearly all the GDB routines and is therefore not mentioned on the diagrams. Whenever an error is detected by a GDB subroutine (ICOND > 0), it calls EPRINT to print an error message and then returns control to the user.

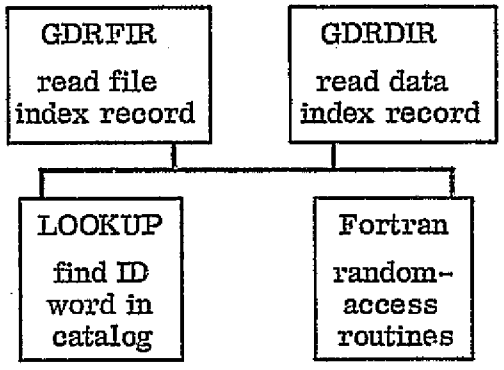

GDOPEN
open
data and
file catalogs
GDADC
attach data
catalog
GDAFC
attach file
catalog
GDATT
attach
permanent file
GDPCAT
print file
or data
catalog
Fortran
random-
access routines
MAKEID
make
ID word

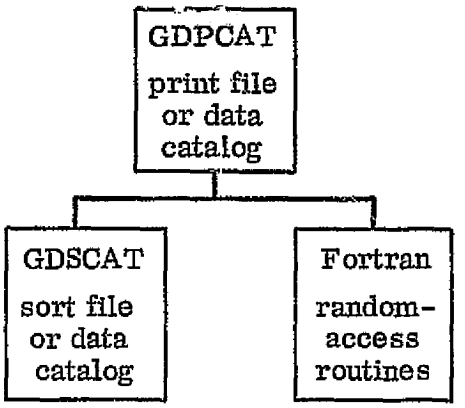

GDADF
attach
data file
GDATT
attach
permanent
file
GDCOPY
disk/tape
copy
PRIMRA
random-
access
package
GDRDIR
read data
index record
GDLABEL
request
tape and check
GDB label
RDBUF
non-Fortran
read/write
routines


GDREAD
read
data record
GDRD
read word
string from
random file
PRIMRA
random-
access
package

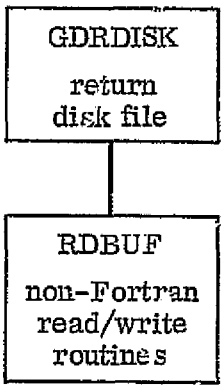

GDWRIT
write a
data record
(sequential)
PRIMRA
random-
access
package
RESERVE
reserve
space on
disk
GDWR
write word
string to
random file
PRIMRA
random-access
package

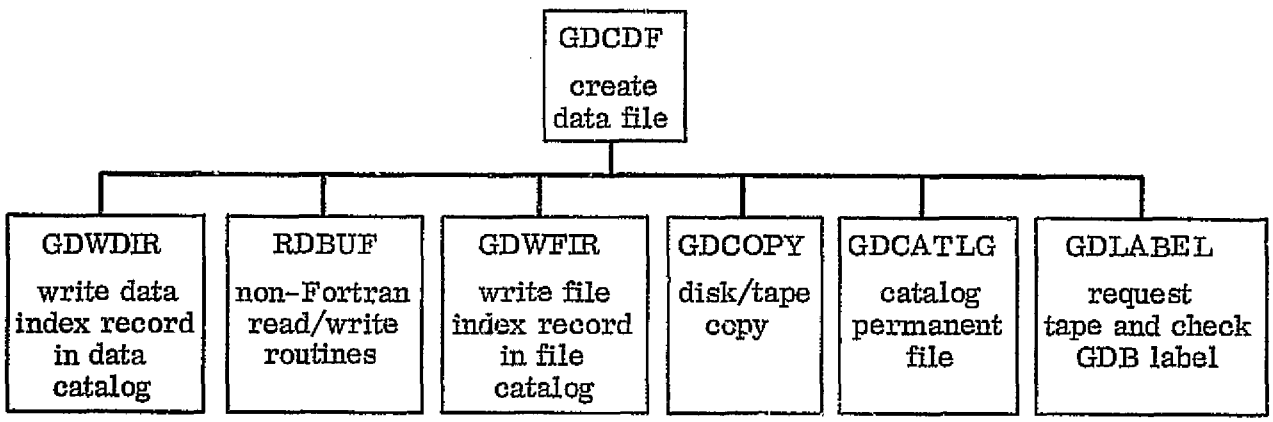

GDREWR
random write data record
PRIMRA
random-access package
RESERVE
reserve space on disk
GDRW
rewrite word string to random file
GDWR
write word string to random file

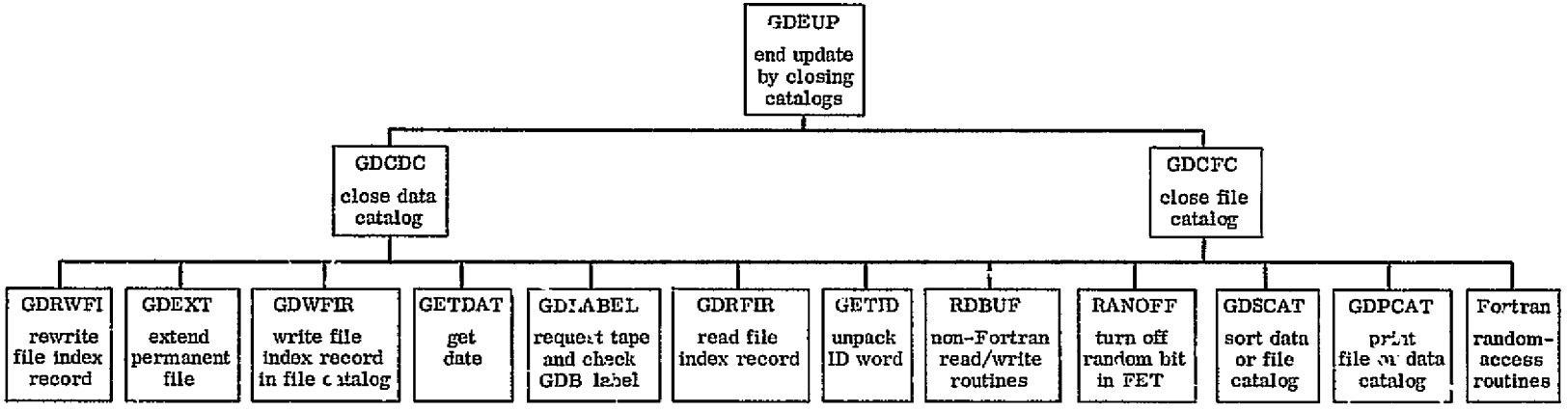

GDRFIR
read file index record
GDRDIR
read data index record
LOOKUP
find ID word in catalog
Fortran
random-access routines

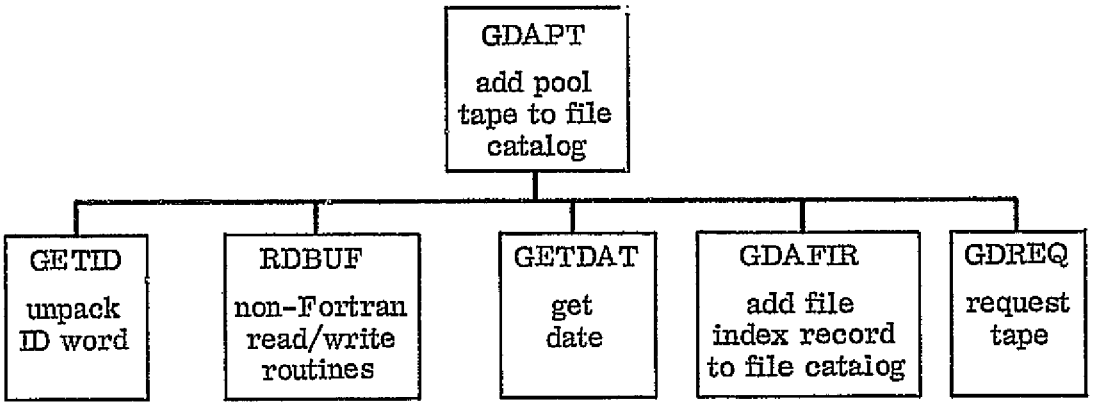

GDPCAT
print file or data catalog
GDSCAT
sort file or data catalog
Fortran
random-access routines

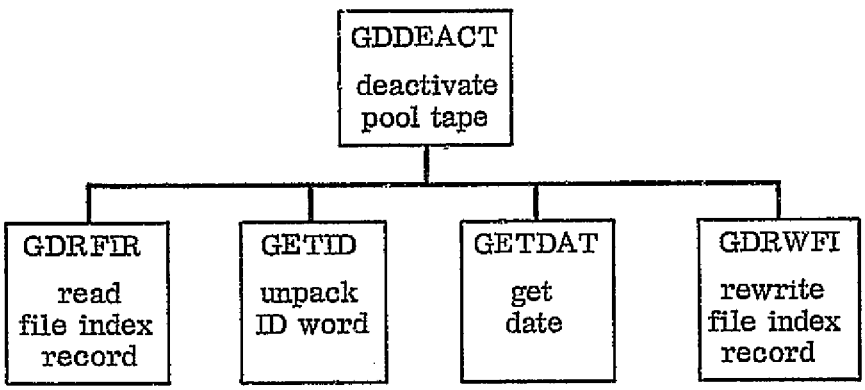

GDRCAT
release catalogs

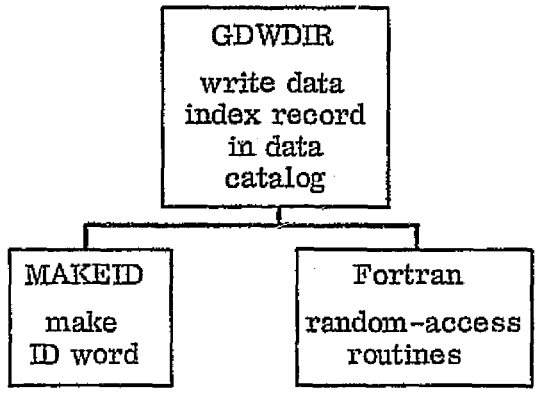

GDRDISK
return
disk file
RDBUF
non-Fortran
read/write
routines

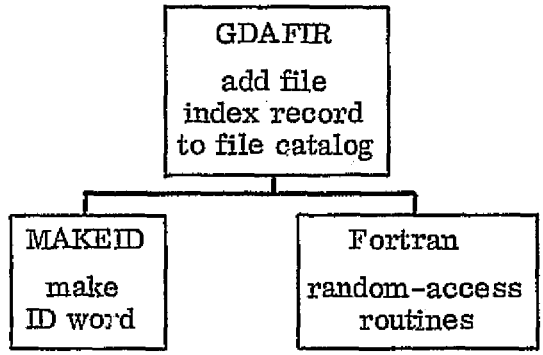

GDCDF
create
data file
GDWDIR
write data
index record
in data
catalog
RDBUF
non-Fortran
read/write
routines
GDWFIR
write file
index record
in file
catalog
GDCOPY
disk/tape
copy
GDCATLG
catalog
permanent
file
GDLABEL
request
tape and check
GDB label

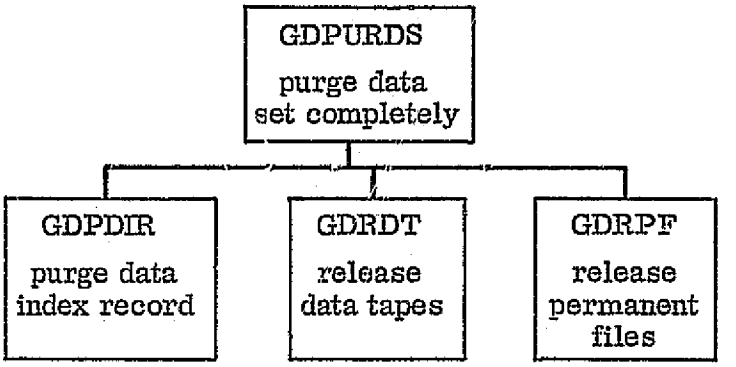

GDEUP
end update by closing catalogs
GDCDC
close data catalog
GDCFC
close file catalog
GDRWFI
rewrite file index record
GDEXT
extend permanent file
GDWFIR
write file index record in file catalog
GETDAT
get date
GDLABEL
request tape and check GDB label
GDRFIR
read file index record
GETID
unpack ID word
RDBUF
non-Fortran read/write routines
RANOFF
turn off random bit in FET
GDSCAT
sort data or file catalog
GDPCAT
print file or data catalog
Fortran
random-access routines

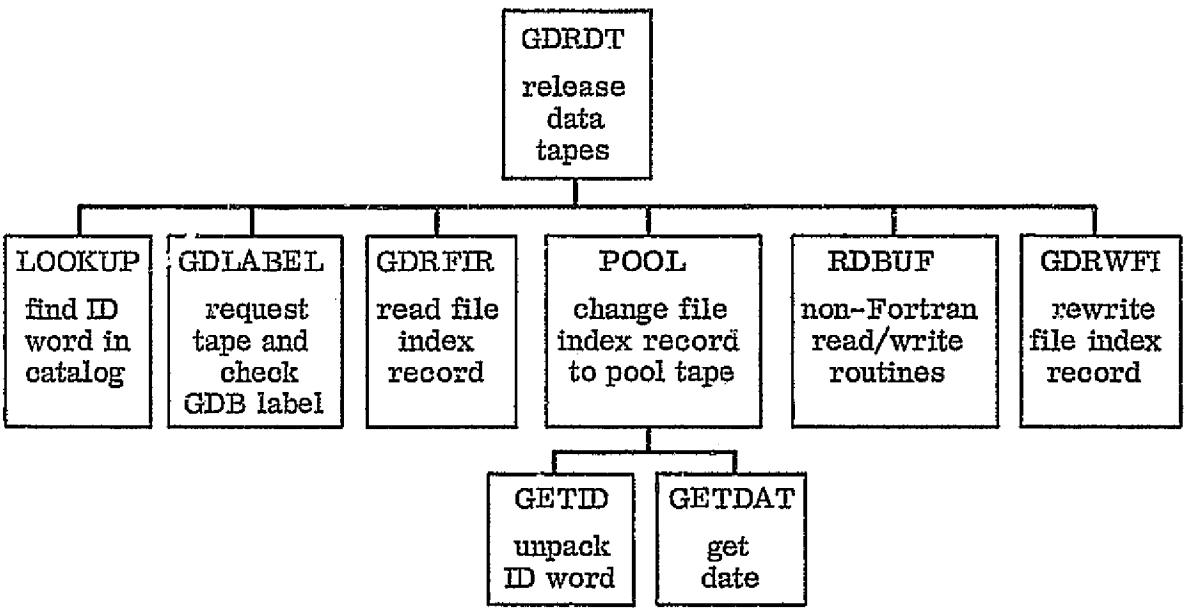

GDAPT
add pool tape to file catalog
GETID
unpack ID word
RDBUF
non-Fortran read/write routines
GETDAT
get date
GDAFIR
add file index record to file catalog
GDREQ
request tape

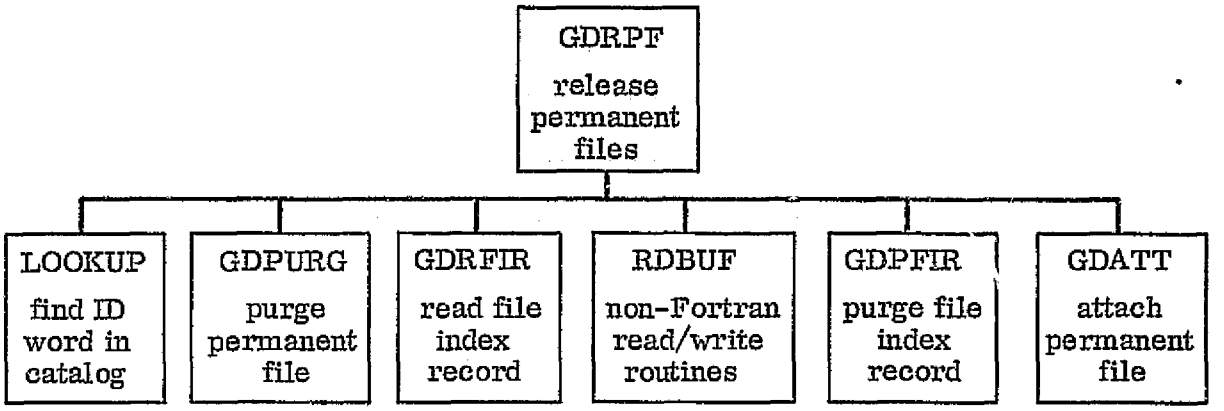

GDDEACT
deactivate pool tape
GDRFIR
read file index record
GETID
unpack ID word
GETDAT
get date
GDRWFI
rewrite file index record

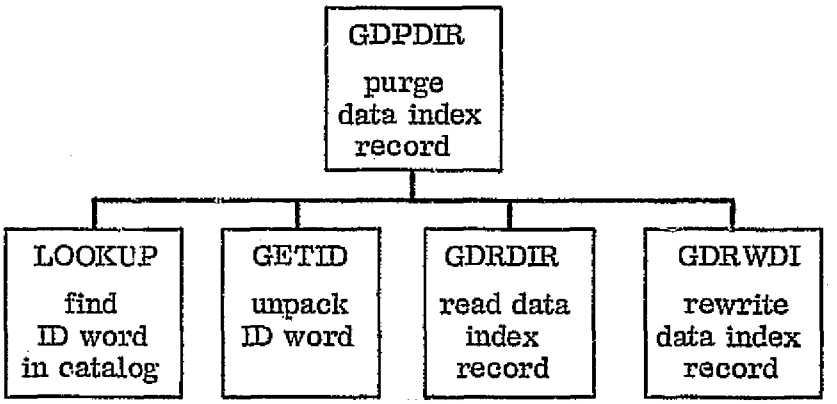

GDWDIR
write data index record in data catalog
MAKEID
make ID word
Fortran
random-access routines

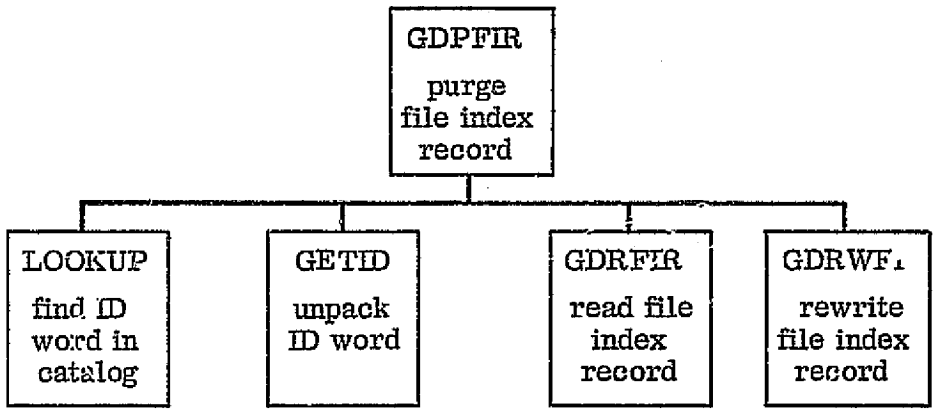

GDAFIR
add file index record to file catalog
MAKEID
make ID word
Fortran
random-access routines

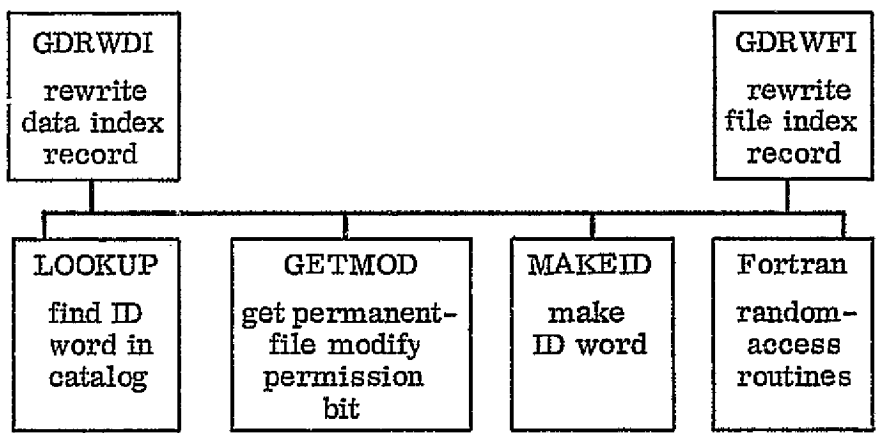

GDPURDS
purge data set completely
GDPDIR
purge data index record
GDRDT
release data tapes
GDRPF
release permanent files

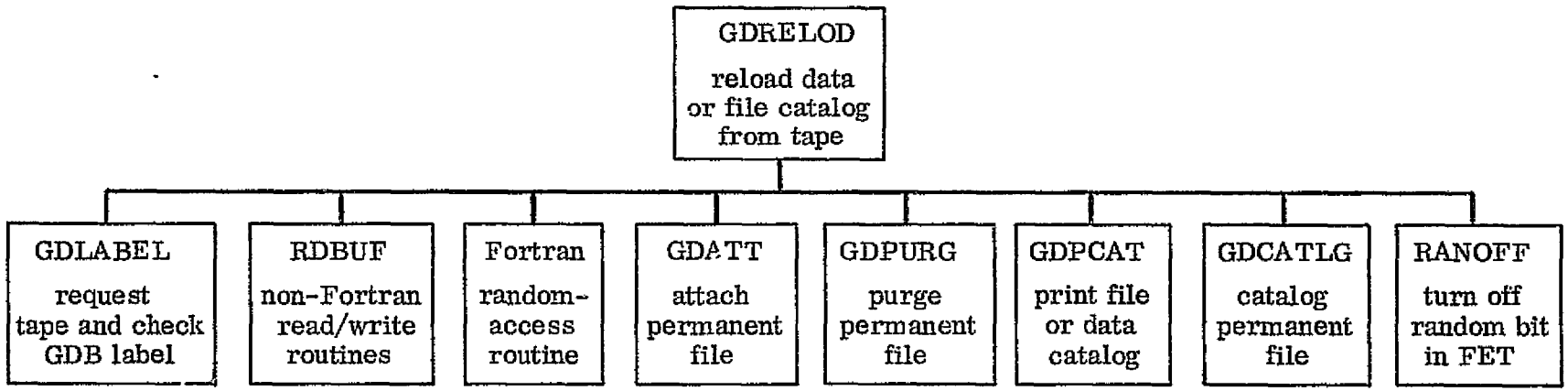

GDRDT
release data tapes
LOOKUP
find ID word in catalog
GDLABEL
request tape and check GDB label
GDRFIR
read file index record
POOL
change file index record to pool tape
RDBUF
non-Fortran read/write routines
GDRWFI
rewrite file index record
GETID
unpack ID word
GETDAT
get date

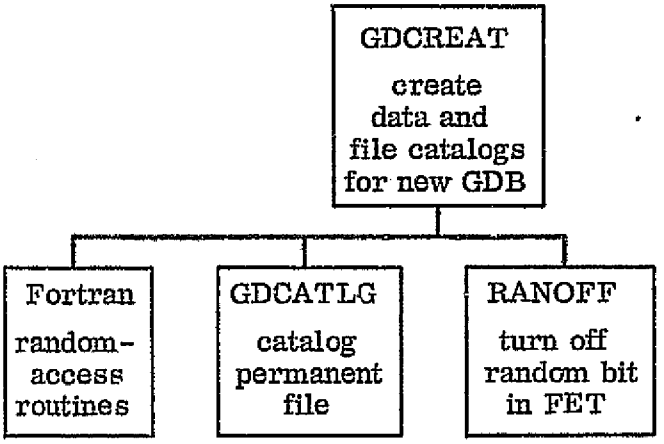

GDRPF
release permanent files
LOOKUP
find ID word in catalog
GDPURG
purge permanent file
GDRFIR
read file index record
RDBUF
non-Fortran read/write routines
GDPFIR
purge file index record
GDATT
attach permanent file

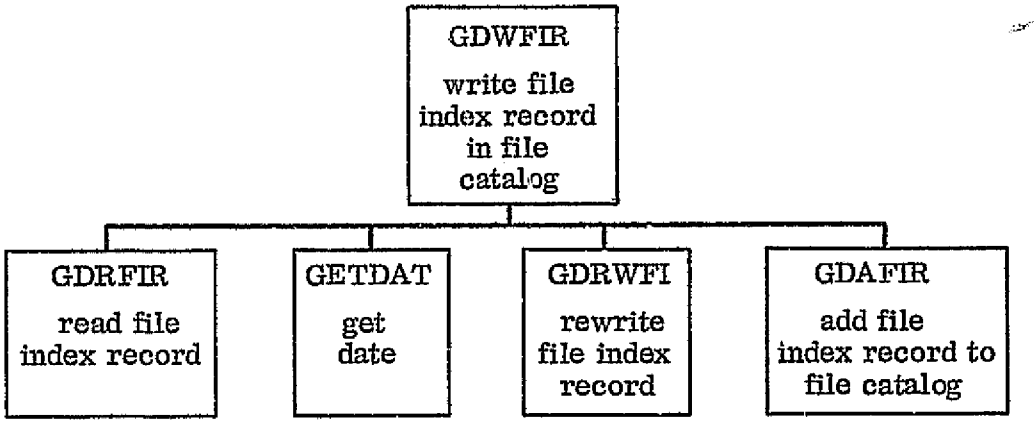

GDPDIR
purge data index record
LOOKUP
find ID word in catalog
GETID
unpack ID word
GDRDIR
read data index record
GDRWDI
rewrite data index record

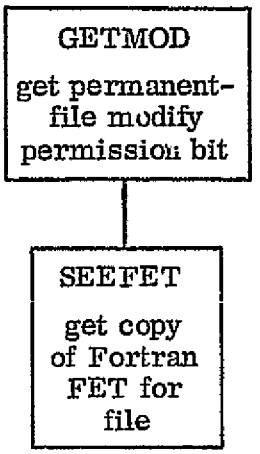

GDPFIR
purge file index record
LOOKUP
find ID word in catalog
GETID
unpack ID word
GDRFIR
read file index record
GDRWFI
rewrite file index record

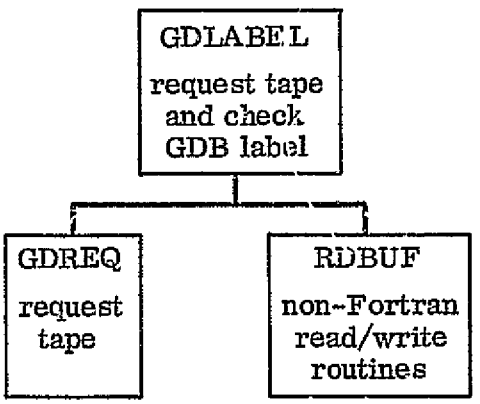

GDRWDI
rewrite data index record
GDRWFI
rewrite file index record
LOOKUP
find ID word in catalog
GETMOD
get permanent-file modify permission bit
MAKEID
make ID word
Fortran
random-access routines

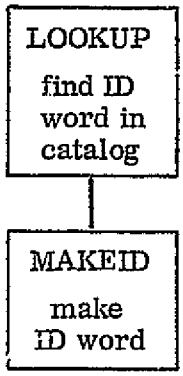

GDRELOD
reload data or file catalog from tape
GDLABEL
request tape and check GDB label
RDBUF
non-Fortran read/write routines
Fortran
random-access routine
GDATT
attach permanent file
GDPURG
purge permanent file
GDPCAT
print file or data catalog
GDCATLG
catalog permanent file
RANOFF
turn off random bit in FET

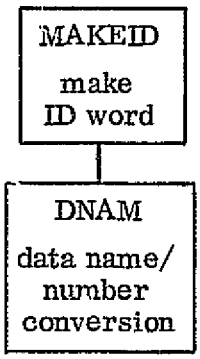

GDCREAT
create data and file catalogs for new GDB
Fortran
random-access routines
GDCATLG
catalog permanent file
RANOFF
turn off random bit in FET


GDWFIR
write file index record in file catalog
GDRFIR
read file index record
GETDAT
get date
GDRWFI
rewrite file index record
GDAFIR
add file index record to file catalog


GETMOD
get permanent-file modify permission bit
SEEFET
get copy of Fortran FET for file


GDLABEL
request tape and check GDB label
GDREQ
request tape
RDBUF
non-Fortran read/write routines


LOOKUP
find ID word in catalog
MAKEID
make ID word


MAKEID
make ID word
DNAM
data name/ number conversion